AN EFFICIENT INTERIOR-POINT DECOMPOSITION ALGORITHM FOR PARALLEL SOLUTION OF LARGE-SCALE NONLINEAR PROBLEMS WITH SIGNIFICANT VARIABLE COUPLING

A Dissertation

by

JIA KANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Arul Jayaraman
Co-Chair of Committee,	Carl D. Laird
Committee Members,	Mahmoud El-Halwagi
	Vivek Sarin
Head of Department,	M. Nazmul Karim

December 2015

Major Subject: Chemical Engineering

ABSTRACT

In this dissertation we develop multiple algorithms for efficient parallel solution of structured nonlinear programming problems by decomposition of the linear augmented system solved at each iteration of a nonlinear interior-point approach. In particular, we address large-scale, block-structured problems with a significant number of complicating, or coupling variables. This structure arises in many important problem classes including multi-scenario optimization, parameter estimation, two-stage stochastic programming, optimal control and power network problems. The structure of these problems induces a block-angular structure in the augmented system, and parallel solution is possible using a Schur-complement decomposition. Three major variants are implemented: a serial, full-space interior-point method, serial and parallel versions of an explicit Schur-complement decomposition, and serial and parallel versions of an implicit PCG-based Schur-complement decomposition. All of these algorithms have been implemented in C++ in an extensible software framework for nonlinear optimization.

The explicit Schur-complement decomposition is typically effective for problems with a few hundred coupling variables. We demonstrate the performance of our implementation on an important problem in optimal power grid operation, the contingency-constrained AC optimal power flow problem. In this dissertation, we present a rectangular IV formulation for the contingency-constrained ACOPF problem and demonstrate that the explicit Schur-complement decomposition can dramatically reduce solution times for a problem with a large number of contingency scenarios. Moreover, a comparison of the explicit Schur-complement decomposition implementation and the Progressive Hedging approach provided by Pyomo is provided, showing that the internal decomposition approach is computationally favorable to the external approach. However, the explicit Schur-complement decomposition approach is not appropriate for problems with a large number of coupling variables because of the high computational cost associated with forming and solving the dense Schur-complement.

We show that this bottleneck can be overcome by solving the Schur-complement equations implicitly using a quasi-Newton preconditioned conjugate gradient method. This new algorithm avoids explicit formation and factorization of the Schur-complement. The computational efficiency of the serial and parallel versions of this algorithm are compared with the serial full-space approach, and the serial and parallel explicit Schur-complement approach on a set of quadratic parameter estimation problems and nonlinear optimization problems. These results show that the PCG implicit Schur-complement approach dramatically reduces the computational expense for problems with many coupling variables.

DEDICATION

This dissertation is lovingly dedicated to my parents: father, Taishan Kang; mother, Qingxia Hao. Their support, encouragement, and constant love have sustained me throughout my life.

ACKNOWLEDGEMENTS

I am so thankful that I am able to write this part of my dissertation now. Undoubtedly, I am a blessed person and I have learned so much and I have been experienced so much. I got my Bachelor Degree in Control Science and Engineering. I have thoroughly pursued the basic knowledge of process control theory, applied that knowledge in the process industries, and obtained extensive experience through industrial projects and auxiliary training. During my six-year pursuing a Ph.D. degree in Dr. Carl Laird's group, I have learned the fundamentals of advanced process modeling, optimization, and numerical computing, focusing on efficient algorithms for nonlinear programming problems and stochastic programming addressing with uncertainties. However, as it comes to real life, there are so many situations that are uncontrollable, full of uncertainties, impossible to see clear and not even mention to optimize. I feel that I have been in this uncomfortable zone for so long that I am finally able to find some way to live with it peacefully. I am able to endeavor with my limited ability and time to do something I think meaningful or right. In all, I would like to thank life for making me a better person, truly.

I would like to thank my advisor, Dr. Carl Laird, for all his guidance and help over the past six years. I miss the time when I was discussing algorithms and implementations with him. I have learned a lot from him and thanks to him, my presentation skills have improved significantly. I also would like to thank Dr. Juergen Hahn for his encouragement. He used to be my co-advisor, but I had to take his name off my committee due to his transferring to Rensselaer Polytechnic Institute. Thanks are also extended to Dr. Arul Jayaraman for his taking care of all the paperwork after Dr. Laird transferred to Purdue University.

Thanks to Dr. Laird's open-mindedness and I did three internships during my PhD study. In summer 2011, I did an internship at the Mathematics and Computer Science (MCS) Division of Argonne National Laboratory. I would like to thank my supervisors, Sven Leyffer and Victor M. Zavala for their inspirational and timely advice. From Oct. 2012 to Feb. 2013, I was invited to do research at Sandia National Laboratories located at Albuquerque, NM. I would also like to thank Jean-Paul Watson for his pleasurable hosting. From Jan. 2014 to May 2014, I did an internship at Mitsubishi Electric Research Laboratories (MERL) in Cambridge, MA. I heartily thank my mentor, Arvind U Raghunathan, for his timely advice and encouragement. I was lucky to meet some fellow friends during those three internships. I would like to thank them for their great companionship. Especially, thank Jiadong Wang for his constant appreciation and support.

Six-years is not a short time. I am blessed to meet so many talented, friendly and interesting people. Thank all the fellow members in Dr. Carl Laird and Dr. Juergen Hahn's group for their help and pleasurable companionship. Particularly, thank Gabriel Hackebeil for his help with Pyomo and thank Yu Zhu for his always comforting and encouraging words. Thank all my fellow graduate students and all the people I met during those six years. I am honored and lucky to meet you all. Thank you for building all my precious memories.

Finally, I would like to thank my parents. Their elaborative inculcation when I was a child, their perpetual love and support helped me step out all the tough situations.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION*

Large-scale, rigorous, nonlinear models are used in many fields of science and engineering to provide accurate simulation of complex systems. As these models are increasingly used in decision making applications, there is new demand for efficient solution approaches to large-scale optimization problems. Coinciding with this need to solve larger nonlinear problems, there has been a change in typical desktop computing hardware. CPU clock rates are no longer increasing significantly, and hardware manufacturers are instead focusing on increasing the number of cores and improving parallel architectures. As the capabilities of a single CPU or workstation are often not enough to tackle emerging problems, it is imperative that we develop effective parallel algorithms to solve these nonlinear optimization problems.

In this section, we will briefly discuss examples of structured nonlinear programming problems (NLP), give an overview of parallel computing architectures, and describe some opportunities for parallel solution of structured NLP problems.

*Part of this section is reprinted with permission from "An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition" by Kang, J., Cao, Y., Word, D., and Laird, C.D., 2014. In: Computers and Chemical Engineering 71 (2014), pp 563-573, Copyright 2014 by Elsevier.

Part of this section is reprinted with permission from "Parallel solution of nonlinear contingency-constrained network problems" by Kang, J., Siirola, J.D., Watson J. and Laird, C.D., 2014. In: Proceedings of the 8th International Conference on Foundations of Computer-Aided Process Design FOCAPD 2014, July 13-17, 2014, Cle Elum, Washington, USA, Copyright 2014 Elsevier B.V.

Part of this section is reprinted with permission from "Nonlinear programming strategies on high-performance computers" by Kang, J., Chiang, N., Laird, C.D., and Zavala, V.M., 2015. In: Proceedings of the 54th IEEE Conference on Decision and Control, 2015, Osaka, Japan, Copyright 2015 Elsevier B.V.

1.1 Nonlinear Programming (NLP) Formulations

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

$$\min f(x) \tag{1.1a}$$

$$\text{s.t. } c(x) = 0 \qquad (\lambda) \tag{1.1b}$$

$$x \geq 0 \qquad (\nu) \tag{1.1c}$$

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

This general problem formulation can be addressed by any number of algorithms, and in this work, we have implemented a primal-dual *interior-point* framework. Nonlinear interior-point methods have proven to be among the most efficient and reliable for this class of problems, and our algorithm is based on the successful open-source project Ipopt (Wächter and Biegler, 2006).

There is an increasing need to address larger and more complex NLP formulations. Examples of this occur as we strive to consider more of the system within a single optimization problem, optimize over discretized systems, or seek to provide treatment of uncertainty within rigorous optimization framework. Fortunately, as problem sizes grow, they are almost always inherently structured, formed from a repeating set of mathematical expressions.

Consider the following nonlinear programming problem, structured with *primal* *coupling*,

$$\min \quad f^d(d) + \sum_{l \in \mathcal{N}} f_l(x_l, d) \tag{1.2a}$$

$$\text{s.t.} \qquad c^d(d) = 0 \tag{1.2b}$$

$$c_l(x_l, d) = 0, \qquad\qquad \forall \; l \in \mathcal{N} \tag{1.2c}$$

$$\underline{x}_l \leq x_l \leq \bar{x}_l \qquad\qquad \forall \; l \in \mathcal{N} \tag{1.2d}$$

$$\underline{d} \leq d \leq \bar{d} \tag{1.2e}$$

where the vector x_l contains all the variables corresponding to one particular block l in the set of blocks $\mathcal{N} = \{1, ..., N\}$. The vectors $\underline{x}_l$ and $\bar{x}_l$ are the lower and upper variable bounds on x_l. The vector d contains the common variables that induce coupling between the blocks. Each block $l \in \mathcal{N}$ would be completely independent if it were not for the coupling variables d (sometimes called common variables or complicating variables). This structure arises in many situations, including optimization under uncertainty, parameter estimation (Zavala et al., 2008; Word et al., 2012), and in stochastic dynamic optimization (Huang and Biegler, 2009; Calafiore and Fagiano, 2013). This structure can also represent problems with spatial decomposition and multi-stage dynamic optimization (where coupling variables for the connections between stages).

Similar problem formulations can be developed for problems with *dual coupling*, where instead of coupling caused by shared variables, coupling is induced by constraints that include variables from more than one block. This structure is common in problems with constraints on shared resources. For example, if $l \in \mathcal{N}$ represents a set of production facilities, and $x_l^{(i)}$ refers to the amount of raw product needed for production facility l, a common constraint might be to restrict the total amount of raw product used (e.g. $\sum_{l \in \mathcal{N}} x_l^{(i)} \leq r$). The primal and dual coupling structures are not completely exclusive, and many problems can be represented in either form. For the purposes of this dissertation, we restrict our discussion to problems with primal coupling only.

Given formulation 1.2, it is often more convenient to introduce dummy coupling variables within each block $l \in \mathcal{N}$, and rewriting the problem as,

$$
\begin{aligned}
\min_{x_l, d} \quad & \sum_{l \in \mathcal{N}} f_l\left(x_l\right) & & \text{(1.3a)} \\
\text{s.t.} \quad & c_l\left(x_l\right) = 0 & \forall \ l \in \mathcal{N} & \quad \text{(1.3b)} \\
& \underline{x}_l \leq x_l \leq \bar{x}_l & \forall \ l \in \mathcal{N} & \quad \text{(1.3c)} \\
& P_l x_l - P_l^d d = 0 & \forall \ l \in \mathcal{N}, & \quad \text{(1.3d)}
\end{aligned}
$$

where d still contains the common variables that induce coupling, however, the elements of d impacting block l are replicated within x_l, and then mapped to d via Equation (1.3d). P_l and P_l^d are mapping matrices. Each x_l in problem 1.3 is now larger than x_l in 1.2 in that it now includes a copy of part or all of the coupling variables d. Furthermore, the interaction between block variables x_l and the coupling variables d occurs linearly, removing the need to compute derivative information for the coupling variables. This is very important for the implementation since it allows us to express the problem using existing modeling languages where each block can be represented as an independent optimization problem (Laird and Biegler, 2008).

Problem 1.3 is remarkably general, and it is the structured NLP that is addressed in the algorithms and applications of this dissertation. The primary goals of this dissertation research are the development of efficient parallel algorithms for tackling this problem, and demonstration of parallel scalability on important applications.

1.2 High-Performance Computing Platforms

Successful development of parallel algorithms to address problem 1.3 (and indeed any problem), requires consideration of the strengths and limitations of the particular parallel computing architecture targeted. Parallel architectures are typically classified according to Flynn's taxonomy, where a key differentiator is whether the architecture can perform different instructions simultaneously.

At one extreme, single-instruction-multiple-data (SIMD) architectures can perform parallel computations; however, each core must be executing the same fundamental instruction simultaneously (albeit on different data). These SIMD architectures are highly appropriate for iterative linear algebra (Cao et al., 2015b) (e.g., PCG, GMRES), but their limitations make it difficult to exploit these architectures for general structural decomposition. Furthermore, while these architectures provide massive parallelism at a relatively low price (e.g. the Tesla K80 provides almost 5,000 cores for a few thousand dollars), they are most effective when the algorithm can be implemented by using a large number of threads to keep the cores loaded (e.g., while waiting for memory operations to complete). Doing so may be difficult in structural decomposition of many large-scale problems. Graphics processing units, commonly used for parallel scientific computing, are a hybrid of the pure SIMD architecture. They comprise a number of multiprocessors, each containing a number of streaming processors or cores. The cores within a single multiprocessor share instructions (i.e. they are true SIMD); and although each multiprocessor can support execution of different kernels, these architectures still do not support parallel execution of different instructions at the same granularity as the number of processing cores. Thus, efficient utilization of these hybrid architectures demands the same considerations as do pure SIMD architectures.

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

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

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

1.3 Parallel Solution of Structured NLP Problems

Parallel strategies for NLP problems can be separated into two broad categories, those algorithms that are inherently parallel on problems of general structure, and those algorithms that are made parallel by exploiting problem specific structure. In this dissertation we focus on the algorithms that exploit problem structure. Within this class of algorithms, there are two broad strategies. Decomposition can be done at the problem formulation level (*external partitioning*), or it can be done internally, at the linear algebra level of a particular NLP algorithm (*internal partitioning*). External approaches (e.g. Bender's decomposition, Lagrangian decomposition, progressive hedging) are far easier to implement, but convergence can be very slow on nonlinear problems. Alternatively, internal partitioning techniques are more intrusive and much more difficult to implement since one must modify all scale-dependent linear algebra operations in the host algorithm. However, this approach retains all the convergence rates and robustness of the host NLP algorithm, and can be much faster than external partitioning techniques.

In this dissertation, we focus on those strategies that achieve parallel speedup by exploiting problem structure and decomposing the internal linear algebra operations performed by the NLP algorithm. While significant work has been done on parallel algorithms for simulation and optimization of partial differential equations with notable codes PETSc (Balay et al., 2014, 2015, 1997) and Trillinos (Heroux et al., 2005), we restrict our discussion to the NLP problem with primal coupling as presented earlier in this section. Furthermore, we focus on the use of nonlinear interior-point methods as the host algorithm of choice. These methods have proven to be among the most competitive serial algorithms for general nonlinear programming problems, and our algorithm is based on the successful open-source project Ipopt (Wächter and Biegler, 2006).

The dominant computational cost for the interior-point methods described in this dissertation is the solution of the so-called augmented system to solve for the step direction at each iteration of the algorithm. Interior-point methods are particularly attractive for development of parallel decomposition strategies because the augmented system retains the same structure for each iteration. Two broad strategies can be used for parallel solution of the augmented system: interface the NLP code with an existing, off-the-shelf parallel linear solver, or write a parallel decomposition approach specifically tailored to the structure of the problem. Several general parallel linear solvers exist, including shared-memory parallel solvers such as PARDISO (Kuzmin et al., 2013; Schenk et al., 2008a, 2007, 2008b) and MA86/MA97 (HSL, 2011) and shared/distributed-memory solvers such as MUMPS (Amestoy et al., 2000, 2001), WSMP (Gupta, 2000), and Elemental (Poulson et al., 2013). Many of these solvers have been used with nonlinear interior-point methods, and Ipopt has existing interfaces to MA86, MA97, MUMPS, PARDISO, and WSMP.

While ease of implementation is a major benefit of using an existing parallel linear solver in one's NLP code, truly scalable performance to hundreds of processors typically requires using a decomposition specifically tailored to the structure of the problem. Amdahl's law provides an estimate of the maximum achievable speedup as the inverse of the fraction of the algorithm that must be executed serially ($S^{\infty}=1/\phi_s$) (Amdahl, 1967). Therefore, in order to achieve significant speedup on large computing clusters, scale-dependent operations of the host algorithm must be serialized. These include model evaluations (which can be parallelized at a block level), and all *vector*, *vector-vector*, and *matrix-vector* operations. For the block structures described in this section, parallel evaluation of the scale-dependent operations is relatively straightforward for all but the solution of the linear system used to compute the step.

Utilizing the techniques outlined in Section 3, parallel decomposition algorithms can be implemented to exploit the specialized block structure in the linear system. At the core of this decomposition approach, the implementation makes parallel calls to separate instances of a serial linear solver for individual blocks (which themselves have the same structure as (2.7)). MA27 and MA57 from the Harwell Subroutine Library (HSL, 2011) have been widely used in serial nonlinear interior-point algorithms and for block decomposition in parallel interior-point methods (Zavala et al., 2008; Kang et al., 2014; Word et al., 2014; Chiang et al., 2014). Of course, many of the parallel linear solvers discussed above perform well in serial, and can be used in this context as well.

Several nonlinear interior-point algorithms have been developed based on structural decomposition of the linear algebra, including OOPS (Gondzio and Grothey, 2007, 2009), PIPS-NLP (Lubin et al., 2011; Chiang et al., 2014), PRBLOCK_IP (Castro, 2007) and Schur-IPOPT (Zavala et al., 2008; Kang et al., 2014). In Castro (2007), structured convex QP problems with constraint coupling are solved through a method that performs Cholesky factorizations on the diagonal blocks and a PCG method for the linking constraints. An explicit Schur-complement approach based on the Ipopt algorithm is implemented in Zavala et al. (2008). This algorithm is appropriate for problems with mild primal coupling; however, the performance deteriorates significantly as the number of coupling variables increases. This is due to the explicit formation of the Schur complement through repeated backsolves and the direct factorization of the dense Schur complement. This work is extended by Kang et al. (2014), using a PCG approach on the implicit equation for the Schur complement. This approach avoids the need to form and factorize the Schur complement, however, it is not appropriate for all the structures described in this section. The PIPS and PIPS-NLP codes implement a number of improvements over standard algorithms, including the use of factorizations of Schur matrix in place of repeated backsolves with columns from B_p^T (Petra et al., 2014), support for recursive block structures, parallel dense factorization of the Schur complement (Lubin et al., 2012), and iterative methods on the Schur complement (Petra and Anitescu, 2012). Another recent code, IPCLUSTER (Cao et al., 2015a), implements an interior-point method for stochastic programming problems that improves the computational time by constructing a sparse, compressed representation of the structured KKT system to compute the step in the coupling variables.

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

Furthermore, for many large-scale problems, construction of the entire model on a serial machine is not possible because of memory and time limitations. Therefore, these languages must support parallel instantiation of partial models along with appropriate metadata to describe this structure to the solver. For many problems, the modeler is aware of the structure and can provide guidance on the construction and labeling. Several languages support suffixes as a mechanism for assigning metadata to variables and constraints, including AMPL (Fourer et al., 1993). This mechanism was used in Zavala et al. (2008) and Kang et al. (2014) to describe coupling where each block in the problem was coded as a separate AMPL model and several instances of the AMPL Solver Library (ASL) were used to support parallel evaluation of the NLP residuals and gradients. Recent work has sought to simplify this effort through the development of modeling languages that provide native support for interfacing with parallel solvers.

Pyomo (Hart et al., 2011) is a python-based open-source algebraic modeling language that supports the definition and solution of optimization applications using the Python scripting language. It is portable and can be used on most platforms. Pyomo supports the general concept of model blocks and allows for custom modeling extensions. The PySP framework (based on Pyomo) provides a high-level interface for parallel instantiation and evaluation of block-structured stochastic programming problems, including interfaces to parallel decomposition algorithms. The structure-conveying modeling language (SML) proposed by Colombo et al. (2009) provides an extension to the AMPL modeling language to support the concept of blocks. A model generation package has been developed for SML that supports parallel instantiation of models described by the block structure in SML (Grothey and Qiang, 2013). Based on the Julia programming language, JuMP (Lubin and Dunning, 2015) provides a mathematical programming modeling language that has compilation and execution speeds similar to those of AMPL, while retaining much of the flexibility of traditional scripting languages. StochJuMP (Huchette et al., 2014) provides an extension to JuMP to support parallel model construction and evaluation for stochastic programming problems. These new developments in modeling languages open the door for mainstream use of specialized parallel solvers.

1.4 Dissertation Outline

In this dissertation work, we have implemented five variations of the nonlinear interior-point method with different strategies for solution of the structured linear system at each iteration. We have chosen to organize the dissertation with the necessary algorithm descriptions and theory first, followed by a discussion of the software implementation, and finally a treatment of the numerical timing results. This dissertation is organized as follows. The nonlinear interior-point method that serves as the base algorithm for the decomposition approaches is shown in Section 2. Different parallel solution strategies will be discussed in Section 3, focusing on the explicit Schur-complement decomposition and our new implicit PCG Schur-complement approach. Section 4 describes the software implementation of the nonlinear interior-point framework and these parallel algorithms. Providing numerical results, parallel solution of the contingency constrained ACOPF Problem will be shown in Section 5 . Sections 6 and 7 demonstrate the performance of our new implicit approach on large-scale problems with significant coupling. Finally, in Section 8 we discuss the significance of our research and propose some future work.

2. INTERIOR-POINT ALGORITHM FOR NONLINEAR PROGRAMMING*

In this section, we discuss the theory and algorithm details for the nonlinear interior-point method implemented in our work. This algorithm and the C++ implementation formed the base algorithm (also called the host algorithm) for the parallel decomposition strategies developed in this dissertation. Section 3 describes the parallel decomposition algorithms described as part of this work, and Section 4 describes the software implementation details for all the algorithms.

First, we will describe the necessary algorithm theory for nonlinear interior-point methods. Consider the general form of an NLP problem with n variables and m nonlinear equality constraints,

$$\begin{aligned} \min\ & f(x) \\ \text{s.t.}\ & c(x) = 0 \\ \underline{x} \leq\ & x \leq \bar{x}, \end{aligned} \tag{2.1}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $c : \mathbb{R}^n \rightarrow \mathbb{R}^m$ are assumed to have continuous first and second derivatives, and $x \in \mathbb{R}^n$. The vectors $\underline{x}$ and $\bar{x}$ are the set of lower and upper variable bounds for x. We solve this problem using an interior-point method with a filter-based line-search based on that described in Wächter and Biegler (2006), and a detailed description of the algorithm and its derivation can be found there. Here, we reiterate only the basic steps necessary to describe our parallel decomposition approach. The barrier subproblem is formed by removing the variable bounds and

*Part of this section is reprinted with permission from "An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition" by Kang, J., Cao, Y., Word, D., and Laird, C.D., 2014. In: Computers and Chemical Engineering 71 (2014), pp 563-573, Copyright 2014 by Elsevier.

adding a log penalty to the objective,

$$\begin{aligned} \min \quad & f(x) - \mu \sum_{i=1}^{n} \ln(\bar{x}^{(i)} - x^{(i)}) - \mu \sum_{i=1}^{n} \ln(x^{(i)} - \underline{x}^{(i)}) \\ \text{s.t.} \quad & c(x) = 0, \end{aligned} \tag{2.2}$$

where μ is the barrier parameter for a single barrier iteration, and (i) denotes the i^{th} element of the vectors of length n.

2.1 First-order Optimality Conditions

The Lagrangian of the optimization formulation (2.2) can then be written as,

$$\mathcal{L} = f(x) - \mu \sum_{i=1}^{n} \ln(\bar{x}^{(i)} - x^{(i)}) - \mu \sum_{j=1}^{n} \ln(x^{(j)} - \underline{x}^{(j)}) + \lambda^T c(x), \tag{2.3}$$

where λ is the vector of equality constraint multipliers. The general optimality conditions are then,

$$\begin{aligned} \nabla_x \mathcal{L} = \nabla_x f(x) + \mu(\bar{G})^{-1} e - \mu(\underline{G})^{-1} e + \nabla_x c(x) \lambda = 0 \\ c(x) = 0, \end{aligned} \tag{2.4}$$

with $\bar{G}$=diag($\bar{x}$−x) and $\underline{G}$=diag(x−$\underline{x}$). The primal-dual formulation is formed by introducing new variables, $\bar{z}$=$\mu[\bar{G}]^{-1}e$ and $\underline{z}$=$\mu[\underline{G}]^{-1}e$. Since the algorithm must maintain $\bar{x}-x \geq 0$ and $x-\underline{x} \geq 0$ (i.e. the points must remain in the interior), it follows that the new variables $\bar{z}, \underline{z} \geq 0$. The addition of these new variables gives the following system of equations:

$$\begin{aligned} \nabla_x \mathcal{L} = \nabla_x f(x) + \bar{z} - \underline{z} + \nabla_x c(x)\lambda &= 0 \\ c(x) &= 0 \\ \underline{G}\underline{z} - \mu e &= 0 \\ \bar{G}\bar{z} - \mu e &= 0. \end{aligned} \tag{2.5}$$

It should be noted that these new equations are the same as the first-order optimality conditions of the original problem (2.1), except with the complementarity conditions now relaxed by μ. A modified Newton's method is used to solve these equations for a particular value of the barrier parameter μ.

2.2 Step Direction and Line-Search

The linear system that must be solved for each iteration k of Newton's method is

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}^k & C^k & -I & I \\ (C^k)^T & 0 & 0 & 0 \\ \underline{Z}^k & 0 & \underline{G}^k & 0 \\ -\bar{Z}^k & 0 & 0 & \bar{G}^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta\lambda^k \\ \Delta\underline{z}^k \\ \Delta\bar{z}^k \end{bmatrix} = - \begin{bmatrix} \nabla_x f^k + \bar{z}^k - \underline{z}^k + C^k\lambda \\ c^k \\ \underline{G}^k\underline{z}^k - \mu e \\ \bar{G}^k\bar{z}^k - \mu e \end{bmatrix}, \tag{2.6}$$

where Δx^k, $\Delta\lambda^k$, $\Delta\underline{z}^k$, and $\Delta\bar{z}^k$ are the full steps for each of the respective variables, $\underline{Z}^k = \mathrm{diag}(\underline{z}^k)$, $\bar{Z}^k = \mathrm{diag}(\bar{z}^k)$, $c^k = c(x^k)$, $C^k{=}\nabla_x c(x^k)$, $\nabla_x f^k{=}\nabla_x f(x^k)$, and $\nabla^2_{xx}\mathcal{L}^k{=}\nabla^2_{xx}\mathcal{L}(x^k)$.

A symmetric system, often called the augmented form, is obtained by multiplying the third block row by $(\underline{G}^k)^{-1}$, the fourth block row by $(-\bar{G}^k)^{-1}$, and adding these rows to the first block row, giving

$$\begin{bmatrix} H^k & C^k \\ (C^k)^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \end{bmatrix} = - \begin{bmatrix} \tilde{r}_x^k \\ c^k \end{bmatrix}, \tag{2.7}$$

where,

$$H^k = \nabla_{xx}^2 \mathcal{L}^k + (\underline{G}^k)^{-1} \underline{Z}^k + (\bar{G}^k)^{-1} \bar{Z}^k \tag{2.8}$$

$$\tilde{r}_x^k = \nabla_x f^k + C^k \lambda - (\underline{G}^k)^{-1} \mu e + (\bar{G}^k)^{-1} \mu e. \tag{2.9}$$

This linear system is significantly smaller than (2.6), and it is symmetric, allowing the use of efficient sparse symmetric linear solvers.

The line-search strategy employed in the interior-point algorithm requires that the generated step be a descent direction. This is ensured if the following inertia condition is satisfied on (2.7) (Forsgren et al., 2002),

$$\mathrm{In}(K) = (n, m, 0) \tag{2.10}$$

where

$$K = \begin{bmatrix} H^k & C^k \\ (C^k)^T & 0 \end{bmatrix}. \tag{2.11}$$

Here, n is the number of variables, and m is the number of equality constraints. For a strictly convex problem under suitable constraint qualifications, this inertia condition is always satisfied. However, we wish to use this algorithm for general non-convex NLPs, and we will make use of inertia correction to ensure descent. The modified linear system is,

$$\begin{bmatrix} H^k + \delta_H I & C^k \\ (C^k)^T & -\delta_c I \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \end{bmatrix} = - \begin{bmatrix} \tilde{r}_x^k \\ c^k \end{bmatrix}, \tag{2.12}$$

where, as described later, δ_H and δ_c are chosen to ensure the inertia condition is satisfied. This system is solved at each iteration of the interior-point algorithm to calculate the full step in x and λ. The steps $\Delta \bar{z}$ and $\Delta \underline{z}$ must also be calculated at each iteration. Algebraic manipulation of the third and fourth rows of the original, unpivoted linear system (2.6) gives,

$$\Delta \underline{z}^k = -(\underline{G}^k)^{-1} \underline{Z}^k \Delta x^k - \underline{z}^k + \mu (\underline{G}^k)^{-1} e \tag{2.13}$$

$$\Delta \bar{z}^k = (\bar{G}^k)^{-1} \bar{Z}^k \Delta x^k - \bar{z}^k + \mu (\bar{G}^k)^{-1} e. \tag{2.14}$$

The variable values for the next iteration are determined by,

$$x^{k+1} = x^k + \alpha^k \Delta x^k \tag{2.15}$$

$$\lambda^{k+1} = \lambda^k + \alpha^k \Delta \lambda^k, \tag{2.16}$$

where α^k is the step size determined by an appropriate line-search. Using these steps, the variable values can be updated for the next iteration using,

$$\underline{z}^{k+1} = \underline{z}^k + \underline{\alpha}^k \Delta \underline{z}^k \tag{2.17}$$

$$\bar{z}^{k+1} = \bar{z}^k + \bar{\alpha}^k \Delta \bar{z}^k, \tag{2.18}$$

where $\underline{\alpha}^k$ and $\bar{\alpha}^k$ are step sizes determined using a fraction to the boundary rule (Wächter and Biegler, 2006; Nocedal and Wright, 2006).

Equations (2.12–2.18) describe how the step is calculated at each iteration of the barrier subproblem.

2.3 Overall Algorithm Description

The complete interior-point algorithm is described in Algorithm 1. Our algorithm is a primal-dual interior-point method with a filter-based line-search based on that described in Wächter and Biegler (2006), and only a high-level description is provided here. The error term in the convergence check performed in steps **2** and **3** is calculated

Algorithm 1 : Interior-point Method

```
1. Initialize the algorithm
   Given starting point $(x^0, \lambda^0, \underline{z}^0, \bar{z}^0)$ with $\lambda^0, \underline{z}^0, \bar{z}^0 > 0$; an initial barrier parameter $\mu_0 > 0$; tolerance constants $\epsilon_{tol}, \kappa_\epsilon > 0$; maximum number of iterations $k_{max}$
   Set the iteration index $k \leftarrow 0$
2. Check convergence of the overall NLP
   if $E(x^k, \lambda^k, \underline{z}^k, \bar{z}^k; 0) \leq \epsilon_{tol}$ then exit, solution found.
3. Check convergence of barrier subproblem
   if $E(x^k, \lambda^k, \underline{z}^k, \bar{z}^k; \mu^k) \leq \kappa_\epsilon \mu^k$ then
     Update $\mu^k$ according to equation (7) in Wächter and Biegler (2006)
     Return to step 2
   end if
4. Calculate functions and gradients
   Evaluate $f(x^k)$, $c(x^k)$, $\nabla_x f(x^k)$, $\nabla_x c(x^k)$, and $\nabla^2_{xx}\mathcal{L}(x^k, \lambda^k)$
5. Compute the search direction (full-step)
    5.1 Solve Equation (2.12) for $\Delta x^k$ and $\Delta\lambda^k$, correcting the inertia if necessary
    5.2 Compute $\Delta\underline{z}^k$ and $\Delta\bar{z}^k$ from Equations (2.13) and (2.14)
    5.3 Compute values for $\alpha^k$, $\underline{\alpha}^k$, and $\bar{\alpha}^k$ using fraction-to-the-boundary rule
6. Update $\alpha^k$ using the line-search filter method from Wächter and Biegler (2006)
7. Update iteration variables and continue to next iteration
   Compute $(x^{k+1}, \lambda^{k+1}, \underline{z}^{k+1}, \bar{z}^{k+1})$ using (2.15–2.18)
   Let $\mu^{k+1} \leftarrow \mu^k$ and $k \leftarrow k + 1$
   if $k < k_{\max}$ then exit with error.
   Return to step 3
```

using,

$$
\begin{aligned}
E(x^k, \lambda^k, \underline{z}^k, \bar{z}^k; \mu^k) = \max\{ \parallel \nabla_x f^k + \bar{z}^k - \underline{z}^k + C^k \lambda^k \parallel_\infty,\ \parallel c^k \parallel_\infty, \\
\parallel \underline{G}^k \underline{z}^k - \mu^k e \parallel_\infty,\ \parallel \bar{G}^k \bar{z}^k - \mu^k e \parallel_\infty \}
\end{aligned}
\tag{2.19}
$$

In this algorithm, the two most computationally expensive steps are **4** and **5.1**. In step **4**, the residuals, gradients, and Hessian are calculated. Fortunately, for structured problems like that shown in Equation (1.3), efficient parallel evaluation of these quantities is readily possible. In step **5.1**, the augmented system is solved. If the original NLP is structured, then structure is induced in the augmented system. For the problem described in Equation (1.3), the augmented system has a block-angular form and parallel solution is possible through a Schur-complement decomposition. In the next section, we describe different strategies for solving the augmented system in parallel.

3. PARALLEL ALGORITHMS FOR STRUCTURED NONLINEAR OPTIMIZATION PROBLEMS WITH INTERIOR-POINT METHODS[*]

In the previous section, we described the base nonlinear interior-point algorithm. Interior-point algorithms have been highly effective for solving large-scale nonlinear programming (NLP) problems and are currently considered among the most powerful methods available to tackle these problems (Nocedal and Wright, 2006). However, as problem sizes continue to grow, standard serial algorithms may not be able to solve these problems. Fortunately, large-scale problems are almost always inherently structured, and these structures enable the development and use of parallel decomposition techniques that can accelerate solutions and avoid memory limitations. Partitioning of the problem can be done externally at the problem formulation level or internally at the linear algebra level. While the external approach is less intrusive and easier to implement, convergence rates and robustness are less favorable. On the other hand, the internal approach is more intrusive and harder to implement, but retains favorable convergence properties of the host algorithm employed. In this section, we will discuss two parallel decomposition approaches implemented based on the interior-point algorithm described earlier.

The dominant computational expense in an interior-point based algorithm like that described in Section 2 is the solution of (2.7), the linear set of equations known as the augmented system, arising from the application of a modified Newton's method

[*] Part of this section is reprinted with permission from "An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition" by Kang, J., Cao, Y., Word, D., and Laird, C.D., 2014. In: Computers and Chemical Engineering 71 (2014), pp 563-573, Copyright 2014 by Elsevier.

Part of this section is reprinted with permission from "Nonlinear programming strategies on high-performance computers" by Kang, J., Chiang, N., Laird, C.D., and Zavala, V.M., 2015. In: Proceedings of the 54th IEEE Conference on Decision and Control, 2015, Osaka, Japan, Copyright 2015 Elsevier B.V.

to the optimality conditions. One promising approach for parallel solution of large-scale NLPs is to parallelize the solution of the augmented system along with other linear algebra operations in the algorithm. Parallel solution of linear systems is typically accomplished in one of two ways: parallel iterative solution or structural decomposition.

Interior-point algorithms are possible utilizing iterative linear solvers with the so-called normal equations, the augmented system, and the doubly augmented system (Forsgren et al., 2007; Dollar, 2007). Efficient solution requires suitable preconditioners, and significant research has been conducted in this field (Oliveira and Sorensen, 2005; Dollar, 2007; Bergamaschi et al., 2004). While a truly effective general preconditioner for use in interior-point methods has been elusive, promising results exist for specific problem classes. For example, Biros and Ghattas (2005) propose the Lagrange-Newton-Krylov-Schur (LNKS) method for steady-state PDE-constrained optimization. This method uses Krylov iterations to solve the full space linearized augmented system utilizing an approximate reduced space quasi-Newton preconditioner.

Effective parallelization of iterative linear solvers is possible, even for unstructured systems. However, real, large-scale, nonlinear optimization problems often possess inherent block structure. Example problem classes that exhibit block structure include optimization under uncertainty, parameter estimation, and spatially decomposable systems. Structure in the optimization problem induces inherent structure in the augmented system solved at each iteration of the interior-point algorithm. Several specialized decomposition-based interior-point algorithms have been developed that exploit block-angular or more general block-bordered structure in the solution of the augmented system. Examples include OOPS (Gondzio and Grothey, 2009), PIPS (Petra and Anitescu, 2012; Lubin et al., 2011), Schur-Ipopt (Zavala et al., 2008), and PRBLOCK_IP (Castro, 2007). The work of Zavala et al. (2008) utilizes an explicit Schur-complement decomposition approach to provide efficient solutions to large-scale optimization problems. Built on the Ipopt algorithm, this approach is appropriate for general NLP problems. This algorithm is appropriate for block-structured problems with complicating variables like those arising in parameter estimation and multi-scenario optimization under uncertainty. In this approach, the Schur-complement is formed explicitly, requiring one backsolve of the system for each complicating variables. As the number of complicating variable increases, forming and solving the Schur-complement explicitly is computationally prohibitive. To overcome the factorization cost, the preconditioned Schur-complement method in PIPS solves the Schur-complement system using a Krylov subspace iterative method with a stochastic preconditioner. The algorithm requires the formation of the Schur-complement system and factorization of the preconditioner (Petra and Anitescu, 2012). Castro (2007) solves the primal block-angular problem by performing Cholesky factorizations on the diagonal blocks of the system and using a preconditioned conjugate gradient method with a truncating power series preconditioner for the linking constraints. This algorithm solves convex QP problems with constraint coupling by exploiting structure in the normal equations.

In this section, we describe two decomposition methods that we have implemented as part of this work. The first, the *Explicit Schur Complement* approach is based on the work of Zavala et al. (2008) to solve large-scale nonlinear block-angular problems using a parallel interior-point method with a Schur-complement decomposition. This approach is known, and has been used in optimization for about 10 years, however, no available, open-source software implementations exist. The second approach we have implemented is the *Implicit PCG Schur-Complement* decomposition is a new algorithm developed as part of this dissertation. As indicated above, the explicit Schur-complement algorithm suffers when the number of coupling variables increases because of the cost of forming and solving the Schur-complement. To avoid this computational expense we have developed a new approach that solves the Schur-complement system using an iterative method that only requires matrix vector products across the equation that describes the Schur-complement. This algorithm solves the Schur-complement system using a preconditioned conjugate gradient method with a limited memory quasi-Newton preconditioner generated using information from the previous CG iterations (Morales and Nocedal, 2000). For example problems with a large number of coupling variables, this algorithm gives an order of magnitude decrease in solution time over methods that explicitly form and solve the Schur-complement.

3.1 Explicit Schur-Complement Decomposition

The algorithm description given in the Section 2 was derived based on the general problem from Equation (2.1). While the problem described in Equation (1.3) fits this form, it provides additional structure that can be exploited to allow for parallel solution techniques. Given the problem shown in Equation (1.3), the augmented system from Equation (2.12), can be rearranged into the block-bordered structure (Zavala et al., 2008),

$$\begin{bmatrix} W_1^k & & & & & A_1 \\ & W_2^k & & & & A_2 \\ & & W_3^k & & & A_3 \\ & & & \ddots & & \vdots \\ & & & & W_N^k & A_N \\ A_1^T & A_2^T & A_3^T & \cdots & A_N^T & \delta_H^k I \end{bmatrix} \cdot \begin{bmatrix} \Delta v_1^k \\ \Delta v_2^k \\ \Delta v_3^k \\ \vdots \\ \Delta v_N^k \\ \Delta d^k \end{bmatrix} = \begin{bmatrix} r_1^k \\ r_2^k \\ r_3^k \\ \vdots \\ r_N^k \\ r_d^k \end{bmatrix}, \tag{3.1}$$

where,

$$
\begin{aligned}
(r_l^k)^T &= -\left[\left(\nabla_{x_l}\mathcal{L}_l^k\right)^T, \left(c_l^k\right)^T, \left(P_l x_l^k - P_l^d d^k\right)^T\right] && \forall \; l \in \mathcal{N}, \\
(\Delta v_l^k)^T &= \left[(\Delta x_l^k)^T \; (\Delta \lambda_l^k)^T \; (\Delta \sigma_l^k)^T\right] && \forall \; l \in \mathcal{N}, \\
A_l^T &= \begin{bmatrix} 0 & 0 & -(P_l^d)^T \end{bmatrix} && \forall \; l \in \mathcal{N}, \\
W_l^k &= \begin{bmatrix} H_l^k + \delta_H^k I & C_l^k & P_l^T \\ \left(C_l^k\right)^T & -\delta_c^k I & 0 \\ P_l & 0 & -\delta_c^k I \end{bmatrix} && \forall \; l \in \mathcal{N}, \\
r_d^k &= \sum_{l \in \mathcal{N}} (P_l^d)^T \sigma_l^k && \forall \; l \in \mathcal{N}.
\end{aligned}
$$

Here λ_l and σ_l are the multipliers for the equality constraints given in Equations (1.3b) and (1.3d), $c_l^k{=}c(x_l^k)$, $C_l^k{=}\nabla_{x_l} c(x_l^k)$, $\nabla_{x_l}\mathcal{L}_l^k{=}\nabla_{x_l} f(x_l^k) + \bar{z}_l^k - \underline{z}_l^k + C_l^k \lambda_l^k$, and $H_l^k{=}\nabla^2_{x_l x_l}\mathcal{L}_l^k + (\underline{G}_l^k)^{-1}\underline{Z}_l^k + (\bar{G}_l^k)^{-1}\bar{Z}_l^k$.

Assuming invertibility of the W_l^k matrices, we can eliminate each of the A_l^T matrices from the bottom block row. The resulting Schur-complement system is given by,

$$
S^k \Delta d^k = r_d^k - \sum_{l \in \mathcal{N}} A_l^T \left(W_l^k\right)^{-1} r_l^k, \tag{3.2}
$$

where

$$
S^k = \delta_H^k I - \sum_{l \in \mathcal{N}} A_l^T \left(W_l^k\right)^{-1} A_l. \tag{3.3}
$$

This system can be solved to find the step in the coupling variables Δd^k, and then the steps for the remaining variables Δv_l^k can be found by solving the following systems:

$$W_l^k \Delta v_l^k = r_l^k - A_l \Delta d^k \qquad \forall\, l \in \mathcal{N}. \tag{3.4}$$

As discussed earlier, inertia correction may be necessary for non-convex problems to ensure that the calculated step be a descent direction, and we need to provide an inertia condition equivalent to Equation (2.10) for the decomposed system. To aid this discussion, let

$$K_{\text{full}}^k = \begin{bmatrix} W_1^k & & & & & A_1 \\ & W_2^k & & & & A_2 \\ & & W_3^k & & & A_3 \\ & & & \ddots & & \vdots \\ & & & & W_N^k & A_N \\ A_1^T & A_2^T & A_3^T & \cdots & A_N^T & \delta_H^k I \end{bmatrix}, \tag{3.5}$$

and

$$K_{\ddagger}^k = \begin{bmatrix} (W_1^k)^{-1} & & & & & \\ & (W_2^k)^{-1} & & & & \\ & & (W_3^k)^{-1} & & & \\ & & & \ddots & & \\ & & & & (W_N^k)^{-1} & \\ & & & & & S^k \end{bmatrix}. \tag{3.6}$$

Additionally, let n_l be the number of variables in block l, that is n_l=dim(x_l), and let m_l be the number of equality constraints in block l, given by Equations (1.3b) and (1.3d). Also, let n_d be the number of coupling variables, that is n_d=dim(d). The total number of variables and equality constraints in the problem is then given by the following expressions,

$$n = n_d + \sum_{l \in \mathcal{N}} n_l \tag{3.7}$$

$$m = \sum_{l \in \mathcal{N}} m_l. \tag{3.8}$$

In the decomposition approach we want to ensure that the inertia condition is satisfied for the full augmented system (3.5) using only information available from the individual blocks W_l^k, and the Schur-complement S^k.

Lemma 1. *If every individual block W_l^k satisfies the inertia condition* $\text{In}(W_l^k)$=$(n_l, m_l, 0)$, *then the full space augmented system matrix K_{full}^k satisfies the inertia condition* $\text{In}(K_{\text{full}}^k)$=$(n, m, 0)$ *if and only if the Schur-complement S^k is positive definite.*

Proof. Given that each W_l^k is symmetric and satisfies the given inertia condition, W_l^k is invertible and has the same inertia as $[W_l^k]^{-1}$. Given the following invertible matrix,

$$Q = \begin{bmatrix} W_1^k & & & & & 0 \\ & W_2^k & & & & 0 \\ & & W_3^k & & & 0 \\ & & & \ddots & & \vdots \\ & & & & W_N^k & 0 \\ A_1^T & A_2^T & A_3^T & \cdots & A_N^T & I \end{bmatrix}, \tag{3.9}$$

we can write

$$QK_{\ddagger}^k Q^T = K_{\text{full}}^k, \tag{3.10}$$

and according to Sylvester's law of inertia (Sylvester, 1952), $K_{\ddagger}^{k}$ and K_{full}^{k} have the same inertia. Therefore,

$$\text{In}(K_{\text{full}}^{k}) = \text{In}(K_{\ddagger}^{k}) = \sum_{l\in\mathcal{N}} \text{In}(W_{l}^{k}) + \text{In}(S^{k}). \tag{3.11}$$

Given the inertia of each W_l^k and Equations (3.7), (3.8), and (3.11), the inertia condition on K_{full}^{k} is satisfied if $\text{In}(S^k)$=$(n_d, 0, 0)$ (i.e. S^k is positive definite). Alternatively, if we assume that the full-space augmented matrix, K_{full}^{k} has the correct inertia, then the Schur-complement must be positive definite. □

In Wächter and Biegler (2006), the inertia of the full-space system is ensured through an inertia correction algorithm. The basic approach is outlined as follows. Initially, δ_H and δ_c are set to zero, and the inertia of the augmented system is calculated during the factorization step. If the inertia indicates the presence of zero eigenvalues, then δ_c is set to a small positive number and the system is factorized again. If the number of positive and negative eigenvalues is not correct, δ_H is increased and the system is factorized again. This is repeated until the inertia condition is satisfied, or a maximum allowable value for δ_H is exceeded, exiting with an error.

In the Schur-complement decomposition, the inertia of each block W_l^k is calculated during factorization. If the inertia of any block W_l^k is not satisfied, then the entire system is corrected by adjusting δ_H and δ_c as above. (Note: it may be possible to locally correct each block independently, however, this is not explored here.) If the inertia of each of the W_l^k blocks is correct, but S^k is found to not be positive definite, then δ_H is increased for the entire system and the procedure repeats. This approach mimics the inertia correction algorithm used when the structure is not exploited.

The Schur-complement decomposition described in Equations (3.2–3.4) decouples the individual W_l^k blocks and allows for parallel solution of the augmented system.

In previous work, we implemented an approach that solved this system in three steps (Zhu and Laird, 2008; Zhu et al., 2011). First, the Schur-complement S^k is explicitly formed using Equation (3.3). Second, the (potentially dense) Schur-complement system in Equation (3.2) is solved to find the step in the coupling variables Δd^k. Finally, the steps for the remaining primal-dual variables are found by solving the systems in Equation (3.4). We call this the Explicit Schur-complement approach.

In the Explicit Schur-complement approach, shown in Algorithm 2, the Schur-complement from Equation (3.2) is explicitly computed. To avoid calculating $(W_l^k)^{-1}$, we instead form the Schur-complement column by column for each block l. Using this approach, a total of N factorizations and $N \cdot n_d$ backsolves are required to form the Schur-complement with another N backsolves required to form the right-hand-side of Equation (3.2).

Algorithm 2 Explicit Schur-Complement (Explicit-SC)

```
1. Form the Schur-complement and the right hand side
   1.1 For each block $l \in \mathcal{N}$:
           Factor $W_l$ and correct inertia if necessary
   1.2 Let $S = [\delta_H I]$ and $r_{sc} = r_d$
   1.3 For each block $l \in \mathcal{N}$:
           For each $j^{th}$ column $A_l{}^{<j>}$ in $A_l$:
              Solve the system $W_l \beta = A_l{}^{<j>}$ for $\beta$
              Update $S^{<j>} = S^{<j>} - A_l^T \beta$
   1.4 For each $l \in \mathcal{N}$:
           Solve the system $W_l p_l = r_l$ for $p_l$
           Update $r_{sc} = r_{sc} - A_l^T p_l$
2. Solve the Schur-complement for the steps in the coupling variables
   Solve $S \; \Delta d = r_{sc}$ for $\Delta d$ (e.g. using a dense linear solver from LAPACK)
3. Solve for the steps in the remaining variables
   For each block $l \in \mathcal{N}$:
          Solve $W_l \Delta v_l = r_l - A_l \Delta d$ for $\Delta v_l$
```

The Schur-complement decomposition decouples individual blocks in the linear solve and allows solution in parallel. In particular, the for loops in steps **1.1**, **1.3**, **1.4**, and **3** can all be executed in parallel instead of sequentially. Given N processors, these steps can be computed efficiently in parallel, using one processor for each block $l \in \mathcal{N}$. Further parallelization is possible in step **1.3** by introducing additional processors and solving individual columns $A_l^{<j>}$ in parallel, however, this requires significantly more processors that cannot be easily utilized in other steps. Therefore, in this work, only the aforementioned steps are computed in parallel. Forming the Schur-complement is linear in n_d, and, if n_d is large, this step can still be computationally expensive due to the large number of backsolves required. Once formed, solving the (potentially dense) Schur-complement using a direct factorization approach has computational complexity that it typically cubic in n_d. While the explicit Schur-complement approach is effective when n_d is sufficiently small (e.g. a few hundred in our experience), the computational cost can become prohibitively expensive as the number of coupling variables increases. If the number of coupling variables is large, we propose the use of a preconditioned conjugate gradient approach to solve the system in Equation (3.2) avoiding both the formation and factorization of this Schur-complement. This approach is described in detail in Section 3.2.

3.2 Implicit PCG Schur-Complement Decomposition

The preconditioned conjugate gradient (PCG) method is an effective technique for the solution of a linear system $Ax{=}b$, where A is a symmetric, semi-positive definite matrix. Several excellent references exist for this approach (Shewchuk, 1994; Saad, 2003). When using the PCG approach to solve the Schur-complement system in Equation (3.2), the greatest computational expense is the matrix-vector product with the Schur-complement. This operation can be performed without ever forming the Schur-complement explicitly. To illustrate this, consider the following product of the Schur-complement with a vector u.

$$\left[\delta_H I - \sum_{l \in \mathcal{N}} A_l^T (W_l^k)^{-1} A_l\right] u \tag{3.12}$$

Before the PCG procedure begins, each of the W_l^k blocks are factorized (and corrected to match the inertia condition outlined previously). Then, the matrix-vector product with u can be performed as follows. For each term in the summation, multiply the vector u by the matrix A_l, perform a backsolve with the block matrix W_l^k using $A_l u$ as the right hand side, and finally, multiply this result by A_l^T. Contributions are summed over each of the blocks $l \in \mathcal{N}$ and subtracted from $\delta_H u$. Note that each iteration of the PCG approach requires a single backsolve with the factors of W_l^k. With the explicit Schur-complement approach we require a backsolve of W_l^k for each column in A_l and each block l (i.e. $n_d \times N$ backsolves). In the PCG approach, we require a backsolve of W_l^k for each PCG iteration and each block l. Therefore, if the number of PCG iterations is smaller than n_d, then we can solve the linear system with fewer backsolves than that required by the explicit approach. The (unpreconditioned) conjugate gradient method requires at most n_d steps (in exact arithmetic) for convergence, and in practice the preconditioned conjugate gradient can require significantly fewer steps, making this approach particularly promising. Furthermore, this approach does not require a dense factorization (cubic in n_d) of the Schur-complement.

The PCG approach could be used to converge the Schur-complement system to a tight tolerance, thereby mimicking the interior-point steps produced by the Explicit Schur-complement approach. However, further reduction in computational cost is possible by solving this system only approximately at early iterations when far from the solution. In our implementation, the CG tolerance for the k^{th} interior-point iteration is obtained from

$$\epsilon_{cg}^{k} = \max\{\epsilon_{cg}^{0}\mu^{k}, \epsilon_{cg}^{min}\}. \tag{3.13}$$

Efficient solution using the PCG method requires a suitable preconditioner. Since the Schur-complement is symmetric and positive-definite (or modified to ensure positive definiteness), an obvious choice for a preconditioner might be a modified BFGS update on the Schur-complement. In a typical BFGS approach for nonlinear optimization, the update would occur once per interior-point iteration. This has several drawbacks. First, the BFGS update is a Rank-2 update and it may take many interior-point iterations to produce a sufficient approximation. Furthermore, the Schur-complement changes as the interior-point iterations proceed, and the information in the BFGS update may become outdated. Finally, storing and operating on the full-memory BFGS update may become prohibitively expensive if the number of coupling variables is large.

On the other hand, the automatic preconditioning technique proposed by Morales and Nocedal (2000, 2001), based on a limited-memory BFGS (L-BFGS) update is appropriate. This approach uses information from the CG iterations performed in the previous interior-point iteration, allowing for extra updates with more current information. At interior-point iteration k, the set of correction pairs $\{s^{k,j}, y^{k,j}\}$ are generated for each CG iteration j using,

$$s^{k,j} = \Delta d^{k,j+1} - \Delta d^{k,j} \tag{3.14}$$

$$y^{k,j} = S^{k}\Delta d^{k,j+1} - S^{k}\Delta d^{k,j} = S^{k}s^{k,j}. \tag{3.15}$$

In the next interior-point iteration, these correction pairs are used with the standard L-BFGS approach to provide the matrix-vector product across the preconditioner. We implemented the preconditioner in C++ based on the algorithm described in Morales and Nocedal (2000) and Morales and Nocedal (2001). Typically, the best number of correction pairs to store is problem (or even iteration) dependent, and Morales and Nocedal (2000) provide a strategy to determine an appropriate number of correction pairs to store between 4 and 20. In the initial interior-point iteration, we solve the Schur-complement system using the CG procedure without a preconditioner, storing some or all of the correction pairs generated in the CG iterations. Since the computational cost of the application of the L-BFGS preconditioner is significantly less than the matrix-vector product across the Schur-complement, we store all the pairs, or 50, whichever is smaller.

The complete PCG Schur-complement (PCGSC) algorithm is shown in Algorithm 3 and the related L-BFGS Preconditioning calculation is shown in Algorithm 4.

Algorithm 3 PCG Schur-Complement (PCGSC)

```
1. Form the right hand side of the Schur-complement system
    1.1 For each $l$ in $N$:
        Factor $W_l$ and correct the inertia if necessary
    1.2 Let $r_{sc} = r_d$
    1.3 For each $l$ in $N$:
        Solve the system $W_l p_l = r_l$ for $p_l$
        Update $r_{sc} = r_{sc} - A_l^T p_l$
2. Solve for the steps in the coupling variables using PCG
    Solve $S\,\Delta d = r_{sc}$ for $\Delta d$ using the iterative PCG procedure with the L-BFGS preconditioner, where correction pairs are stored according to Algorithm SAMPLE in paper Morales and Nocedal (2000)
3. Solve for the steps in the remaining variables
    For each $l$ in $N$: solve $W_l \Delta v_l = r_l - A_l \Delta d$ for $\Delta v_l$
```

Algorithm 4 L-BFGS Preconditioning Step

```
Given $M$ correction pairs $(s_i, y_i)$, $\rho_i = \frac{1}{s_i^T y_i}$, to calculate the preconditioner multiplying the current CG residual $r$.
$\gamma^0 = \frac{s_M^T y_M}{y_M^T y_M}$
$q \leftarrow r$
for $i = M, M-1, ..., 2, 1$ do
    $\alpha_i \leftarrow \rho_i s_i^T q$
    $q \leftarrow q - \alpha_i y_i$
end for
$\xi \leftarrow \gamma^0 q$
for $i = 1, 2, ..., M-1, M$ do
    $\beta \leftarrow \rho_i y_i^T \xi$
    $\xi \leftarrow \xi + s_i(\alpha_i - \beta)$
end for
stop with result $\xi$, the preconditioner multiplying the current CG residual $r$
```

Similar to the explicit approach, it is straightforward to perform Algorithm 3 in parallel. Given N processors, the for loops in Steps **1.1**, **1.3**, and **3** can be done in parallel. As well, the matrix-vector product required in step **2** and described in Equation (3.12) can be completed in parallel by using one processor for each term in the summation. If the individual blocks W_l are themselves of prohibitive size, further parallelization is possible using a parallel linear solver for general sparse symmetric matrices, although this hybrid approach is not tested here.

Both the *Explicit Schur-Complement* method and the *Implicit PCG Schur-Complement* method have been implemented within the interior-point framework developed in Section 2. Both these algorithms have been implemented in serial and in parallel. This gives five different algorithms in total. A summary of these five different algorithms is given in the next section.

3.3 Summary of Different Algorithms

Solution of the augmented system (step direction calculation in the flowchart) is the dominant computational expense in every interior-point iteration, and as described in the previous section, the major algorithm variations in our implementation differ in their solution strategy for this linear system. This section describes the implementation of five different algorithm variants, listed as follows:

Full-space Serial (FS-S): The full-space serial option solves the KKT system directly using the MA27 routine from the Harwell Subroutine Library (HSL, 2011). All linear algebra operations are performed in serial.

Explicit Schur-complement Serial/Parallel (ESC-S/ESC-P): This option implements the explicit Schur-complement approach for the solution of the KKT system. In this block decomposition, all W_l^k blocks are factored using MA27 from the Harwell Subroutine Library, and the explicitly formed Schur-complement is solved using the cholesky factorization routine from LAPACK.

PCG Schur-Complement Serial/Parallel (PCGSC-S/PCGSC-P): This option implements the Schur-complement decomposition strategy using the preconditioned conjugate gradient method to solve the Schur-complement system. In this block decomposition, all W_l^k blocks are factored using MA27 from the Harwell Subroutine Library. The Schur-complement is solved implicitly, avoiding the need to form or factorize the Schur-complement directly.

The software implementation details for these algorithms is discussed in the next section, followed by some numerical results comparing the explicit approach with an external decomposition method (Progressive Hedging), and a comparison of the explicit and the implicit approach on different problems.

4. FRAMEWORK AND SOFTWARE IMPLEMENTATION FOR SERIAL AND PARALLEL ALGORITHMS*

In this section, we describe the software implementation details of the different algorithms that were developed as part of this dissertation. All the different algorithms described in the previous section (i.e., FS, ESC-S, ESC-P, PCGSC-S, and PCGSC-P) are implemented within the same software framework, but with different `main` functions and executables. Here, we first describe the basic flowsheet of the algorithm and then describe the software architecture (class hierarchy and inheritance structure) used to support the different algorithms.

All algorithms are based off of the interior-point framework described in Section 2. Figure 4.1 shows the basic flowsheet of the implemented interior-point framework. First, the optimality conditions of the original problem are checked for convergence. If the original NLP problem is not converged, we proceed to check convergence of the barrier NLP subproblem. If the barrier NLP subproblem is also not converged, the derivatives, residuals, etc. are calculated and based on the current point, and the step direction is computed by solving the augmented system (and updating the multipliers). Once the step direction is obtained, the filter line search is used to determine the appropriate step length, and the loop continues until the barrier NLP subproblem converges or maximum iterations are reached. Once the barrier NLP subproblem has converged, then barrier parameter μ is reduced and the whole process is repeated until the original NLP problem converges. Of course, there are far more details in the algorithm this short overview describes.

*Part of this section is reprinted with permission from “An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition” by Kang, J., Cao, Y., Word, D., and Laird, C.D., 2014. In: Computers and Chemical Engineering 71 (2014), pp 563-573, Copyright 2014 by Elsevier.

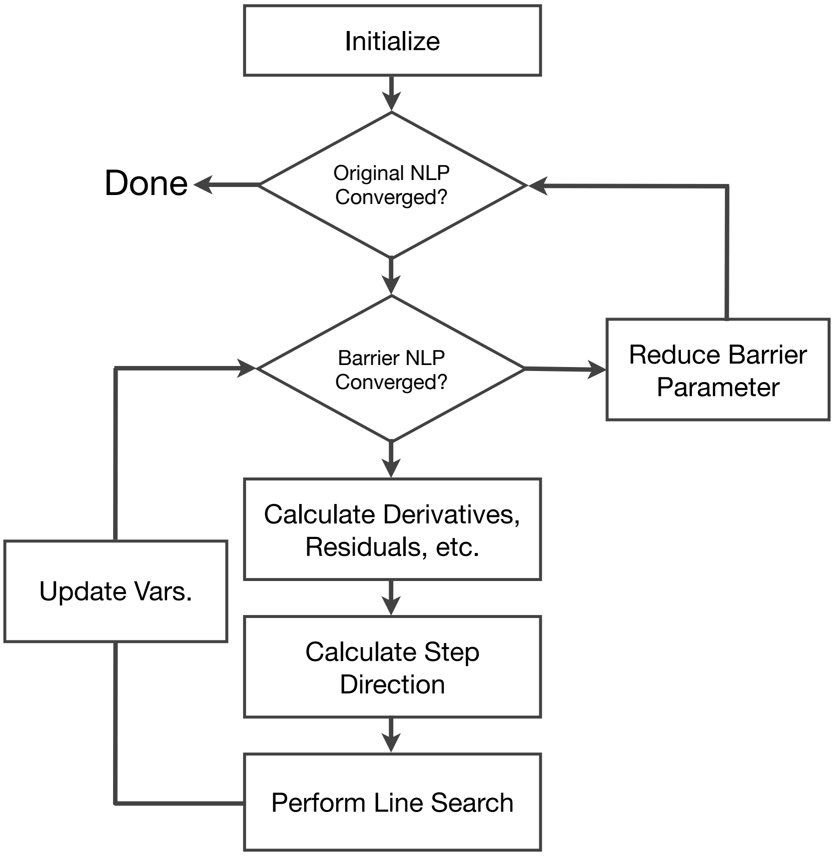


Figure 4.1: Flowchart of the interior-point algorithm.

4.1 High-level Overview of Software Implementation

All the algorithm development was done in C++, and the Message Passing Interface (MPI) used for all parallel communication. An object-oriented approach was used to provide clean separation between the components of the code that were the same across all algorithm variants, and those that were different. The algorithms listed above differ in two fundamental aspects. First, the specific algorithm used to solve the augmented system (2.7) is different across all five variants. As well, the parallel and serial implementations differ for the same algorithm. In addition to the parallel computation or operations within the algorithm, effective scaleup to many processors requires parallel evaluation of the model as well. For this reason, the software implementation for the problem representation (the NLP) has three variants, the serial implementation of the full-space problem, and the serial and parallel versions of the structured problem with primal coupling. Note that the serial structured representation is shared among ESC-S and PCGSC-S, while the parallel structured representation is shared among ESC-P and PCGSC-P.

Although the linear algebra operations and the model evaluation differ across the approaches, the same fundamental interior-point algorithm is used for all five variants. Therefore, it is attractive to use an object-oriented structure that permits re-use of the fundamental algorithm code (the interior-point method) while allowing specialization of the model and linear algebra components. To support this behavior, the interior-point algorithm code was implemented with very strict interfaces between the algorithm, the model interface, and the linear algebra interface. A high-level description of the implementation is shown in Figure 4.2 below.

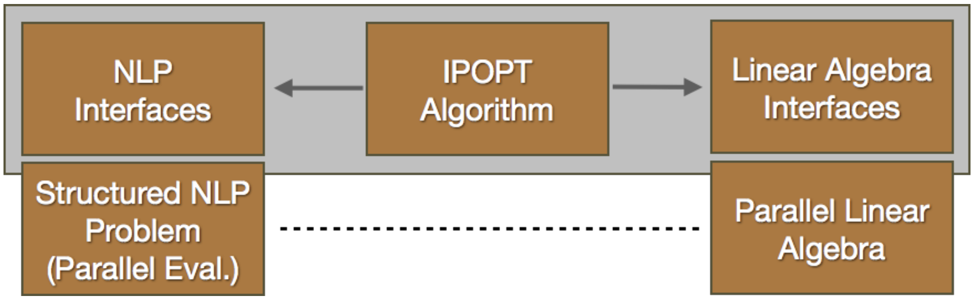

NLP Interfaces
IPOPT Algorithm
Linear Algebra Interfaces
Structured NLP Problem (Parallel Eval.)
Parallel Linear Algebra

Figure 4.2: High-level description of the software structure.

Here, the code for the interior-point algorithm is independent of the specific implementation for the model representation and the linear algebra operations, and it communicates with these components through well-defined interfaces. With this structure, the interior-point algorithm is unaware of the underlying data structure for the linear algebra objects (e.g. it does not have the ability to access individual elements of the vectors and matrices). Rather, it performs all necessary operations through the linear algebra interface (and the NLP interface). Therefore, we can implement specialized NLP representations and specialized linear algebra routines that are aware of the inherent structure of the problem, and both of these specializations can be done without changes to the fundamental interior-point code. This software structure has made it possible to implement several variants in a convenient object-oriented framework. This design also eased the parallelization of these algorithms. Since the individual elements in any matrix or vector are hidden from the fundamental algorithm and only the interfaces are exposed, the underlying algorithms operate identically whether serial or parallel code is being used. Note that the algorithm requires several vector, vector-vector, and matrix-vector linear algebra operations (e.g. dot product, norms, matrix-vector multiplication), and these operations are also parallelized in our implementation. The Message Passing Interface (MPI) is used for all parallel communication.

4.2 Detailed Software Implementation Description

As indicated above, the implementation uses an object-oriented software design. The NLP problem representation is separated from the fundamental algorithm code, and the interior-point code communicates only through a well defined `NLP` base class. Furthermore, while the solution of the augmented system is the dominant computational expense in the algorithm, achieving good parallel performance requires that the NLP evaluations (e.g., residuals, gradients, Hessian) and the other operations of the algorithm (i.e., vector, vector-vector, matrix-vector operations) also be performed in parallel. The implementation details of these linear algebra operations are hidden from the interior-point algorithm, and the algorithm performs these operations through well defined interfaces in the `Matrix`, `Vector`, and `KKTLinearSolver` base classes. The major classes in the implementation are shown in the class hierarchy given below where the base class are left aligned and further derived classes indented (e.g. `FullSpaceLinearSolver` inherits from `KKTLinearSolver`).

Table 4.1: Class Hierarchy

- AmplInterface
- BlockSystem
- DirectedGraph
- InteriorPointSolver
- KKTLinearSolver
  - FullSpaceLinearSolver
  - MPISchurComplementLinearSolver
  - MPIStrongSchurComplementLinearSolver
  - SchurComplementLinearSolver
- MA27_LinearSolver
- MA48_LinearSolver

- MA57_LinearSolver
- Matrix
  - BlockMatrix
  - CoordMatrix
    - SymCoordMatrix
  - DenseMatrix
  - DiagMatrix
  - MPIBlockDiagMatrix
  - MPIBlockMatrix
  - MPISymBlockMatrix
  - MPIVarCoupledHessian
  - MPIVarCoupledJacobian
  - SymBlockMatrix
  - SymDenseMatrix
  - SymFullDenseMatrix
- MPIWrapper
- NLP
  - AmplNLP
  - MPIVarCoupledBlockNLP
  - MPIVarCoupledMultiBlockNLP
  - VarCoupledBlockNLP

- PARDISO_LinearSolver
- Preqn
- PreqnEntry
- PreqnSample
- PreqnSampleEntry
- SUPERLU_LinearSolver
- TaskTimer
- UMFPACK_LinearSolver
- UnreducedInteriorPointSolver
- Vector
    - BlockVector
    - DenseVector
    - MPIBlockVector

There are many classes and over 12,000 lines of C++ code within this implementation. In the following text, we will briefly describe the key classes used in the implementation.

It should also be noted that some of this code is part of a set of common code that is used by the Laird research group for different projects. Some of the common code was developed as part of this research dissertation work, and some was developed as part of other research projects. Of course, classes specific to the parallel decomposition algorithms and problem representations are from this dissertation work.

4.2.1 `InteriorPointSolver`

The main workhorse of the algorithms is the interior-point implementation contained in the `InteriorPointSolver` class. While this class implements the main flowsheet for the interior-point algorithm discussed in Section 2, it makes use of several other classes to perform the necessary computations. It performs all linear algebra operations through the `Matrix` and `Vector` base classes. This means that the interior-point algorithm code does not have access to the specific derived implementations of these classes. All necessary linear algebra operations must be available on these base classes, or composable through multiple operations on these base classes. Of course, specific derived implementations of these classes are given to the `InteriorPointSolver` (through the `NLP` interface), but it does not know about these further derived details. This allows the underlying operations to be specialized without changes in the main solver code. In a similar fashion, when the `InteriorPointSolver` requires a step computation, this is done through the `KKTLinearSolver` base class, and any interactions with the problem representation (e.g. compute residuals, Jacobians, etc.) is done through the `NLP` base class. The entire interaction is driven by main when the problem is to be solved, the `InteriorPointSolver::Solve` method is called and passed a specific instantiation of the `NLP` class and the `KKTLinearSolver` class.

4.2.2 `Matrix` and `Vector` and Derived Classes

As indicated above, the `InteriorPointSolver` class performs all required linear algebra operations through a set of defined interfaces in a few base classes. Two key base classes are the Matrix and Vector classes. These classes provide the basic interface for working with Hessians, Jacobians, and other matrices and vectors. The class diagram for the Vector class is shown in Figure 4.3

The class diagram for the Matrix class is shown in Figure 4.4. In both the hierarchies, we have implemented several specialized classes for different structures. Of particular note are the block matrices and MPI block matrices that implement a general block structured matrix where other matrix subclasses can be pieced together to form larger systems without the need for copying data (or even having data reside within the same process).

4.2.3 KKTLinearSolver and Derived Classes

The KKTLinearSolver provides the base class interface for performing a step computation. The class diagram showing the inheritance structure and class members for KKTLinearSolver and derived classes is shown in Figure 4.5. The primary method on this class is the pure virtual method `KKTLinearSolver::Solve(Matrix &KKT, Vector &diag, Vector &rhs, Vector &soln, const int inner_iter, const double mu)` that must be overridden by the derived classes. Again, the InteriorPointSolver class interacts directly with a KKTLinearSolver object that is actually one of the further derived classes:

- FullSpaceLinearSolver: This class implements the standard full space nonlinear interior point approach that makes use of a symmetric indefinite linear solver (typically based on a Bunch-Kaufman algorithm as to provide the inertia evaluation inexpensively). In this implementation, MA27/MA57 has been used, however, interfaces exist to other appropriate linear solvers.

- SchurComplementLinearSolver: This class implements the serial explicit Schur-complement approach (ESC-S) and the serial implicit PCG Schur-complement

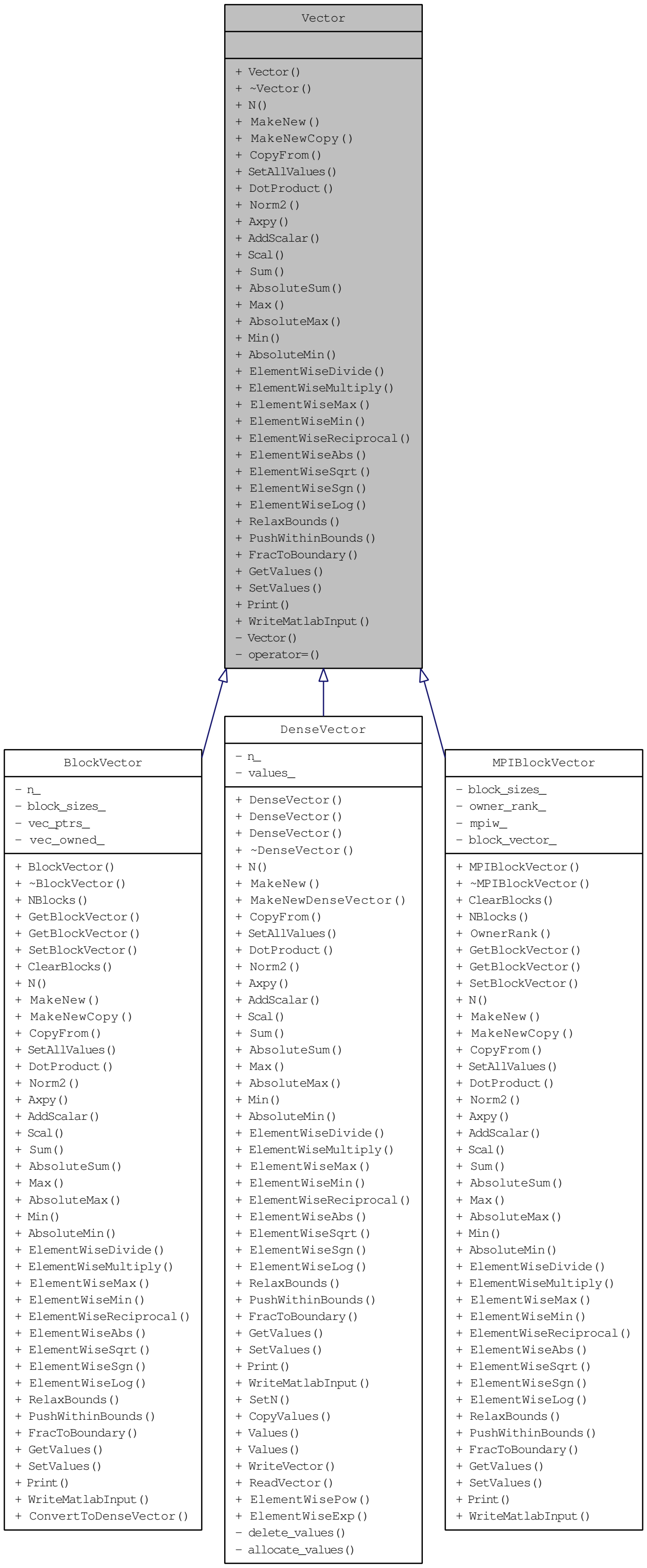


Figure 4.3: Class diagram for `Vector`

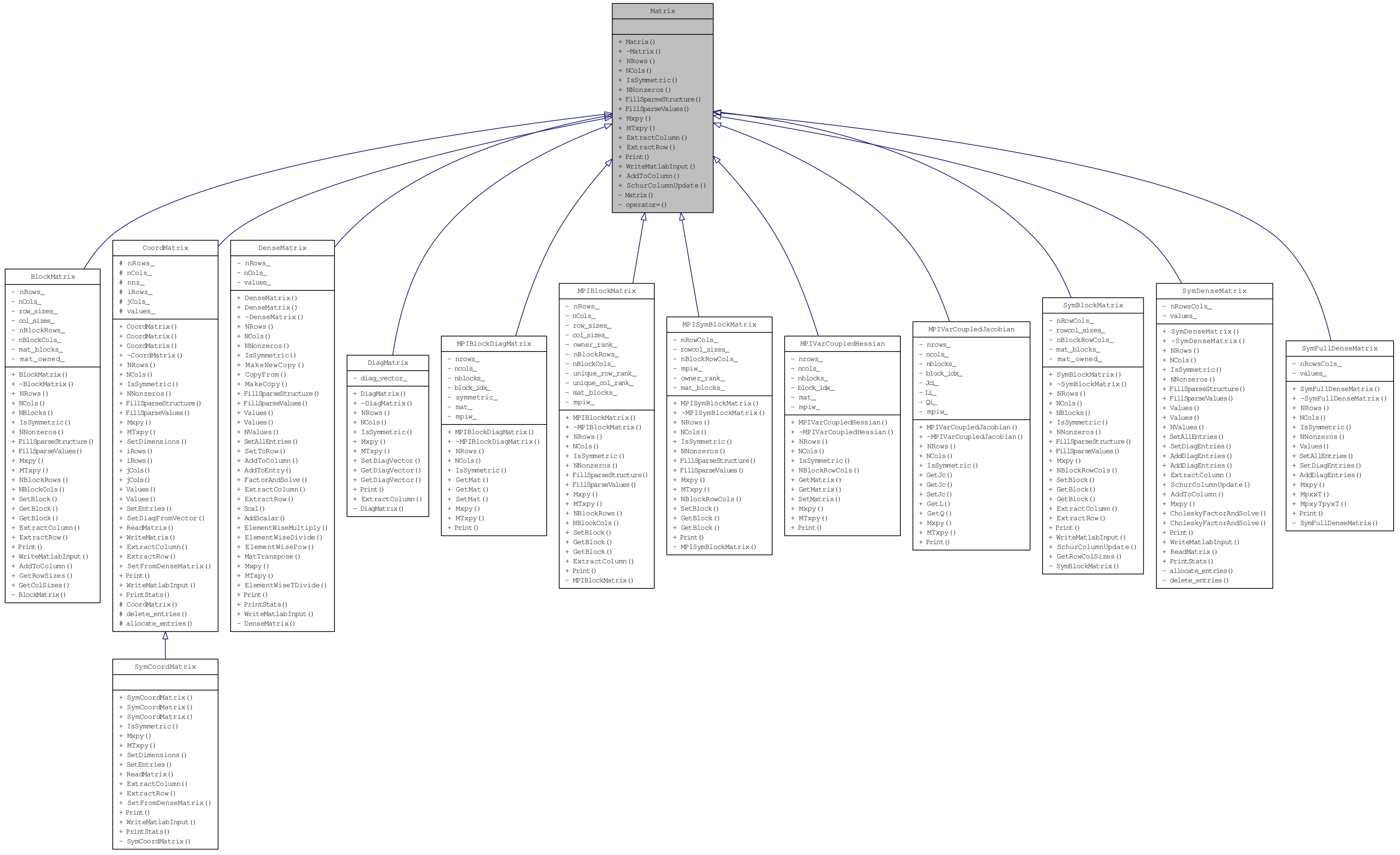


Figure 4.4: Class diagram for `Matrix`

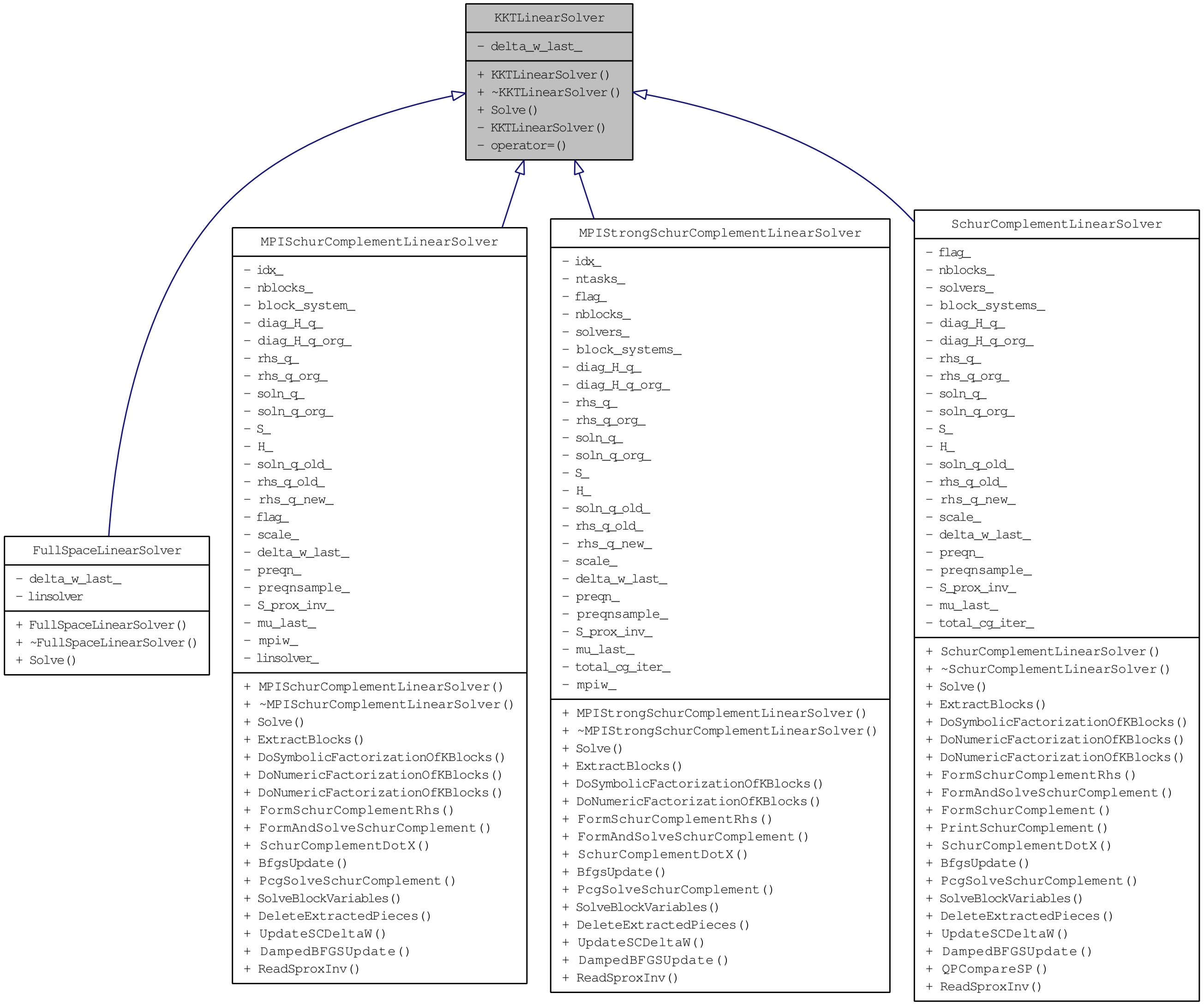


Figure 4.5: Class diagram for `KKTLinearSolver`

approach (PCGSC-S). MA27/MA57 is used to solve each of the individual W_L blocks, and LAPACK is used to solve the Schur-complement.

- MPISchurComplementLinearSolver, MPIStrongSchurComplementLinearSolver: These classes implement the parallel Schur-complement approaches (ESC-P) and (PCGSC-P). These two classes differ in the way they handle the decomposition when the number of processors is smaller than the number of blocks. In one case, the blocks are aggregated, and the algorithm sees the same number of blocks as processors. In the other case, the blocks are still divided as small as possible, and one process handles several blocks (much like the serial approach).

Since these classes implement the primary parallel effort in this dissertation, further discussion is given on their implementation. Figures 4.6 through 4.8, shows the structure of the KKT matrix for a 4-block problem. The original structure is shown in Figure 4.6, where its hessian is a diagonal block matrix and its Jacobian is also block structured, where each block is coupled by the terms corresponding to the common variables. This KKT matrix can be arranged using some row and column permutation to generate an arrowhead structure, shown in Figure 4.7. Much of the structure within the blocks is ignored in the mathematical representation, and the system is usually written as shown in Figure 4.8. It should be noted, however, that both the W_l and A_l blocks have additional structure that can be seen in Figure 4.7 and is used in the implementations.

It should also be noted that each process only owns the portions of the block structure that are necessary to perform their portion of the computations. No single process owns the full KKT system. To see how this is done, consider the parallel explicit Schur-complement approach (ESC-P). Recall the expression for the Schur-

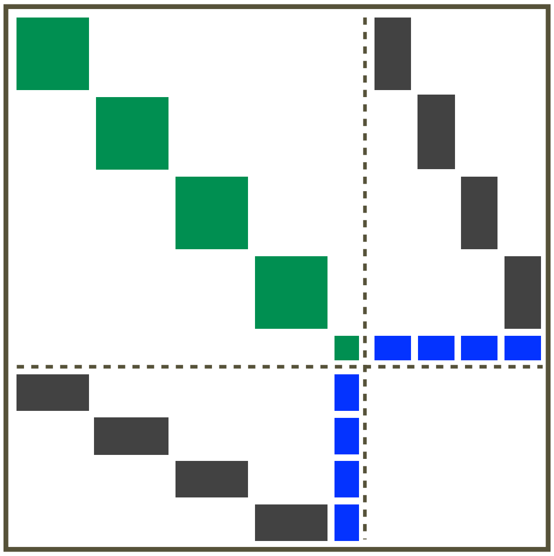


Figure 4.6: Original KKT matrix structure for a 4-block problem

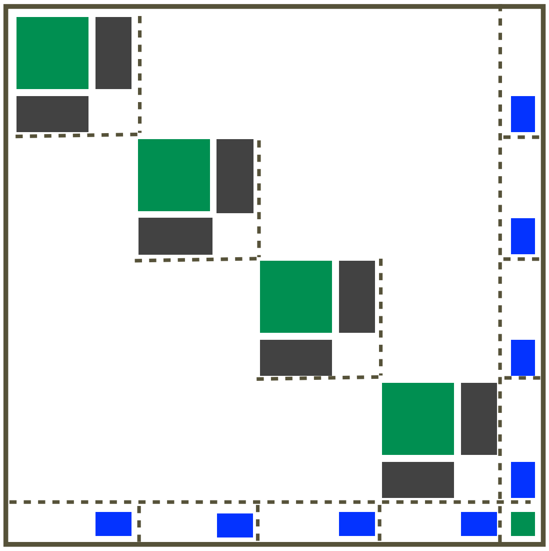


Figure 4.7: Permuted KKT matrix structure for a 4-block problem

complement,

$$S^k = \delta_H^k I - \sum_{l \in \mathcal{N}} A_l^T \left(W_l^k\right)^{-1} A_l.$$

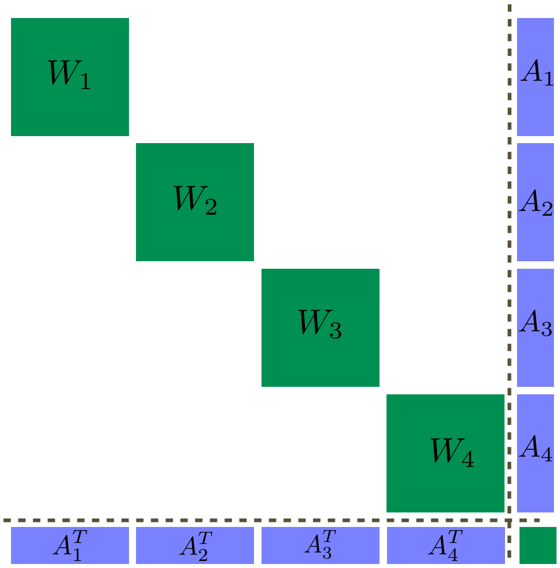


Figure 4.8: Permuted KKT matrix structure for a 4-block problem with mathematical denotation

If we have the same number of processors as we do blocks, then each processor can form one element of the summation. We can see then that each processor l only needs access to A_l and W_l. Thus, the KKT matrix can be stored distributed, as shown in Figure 4.9. Although not shown in this diagram, we can, of course, assign one or multiple blocks to one processor. This process ownership is both effective and convenient, since individual NLP objects can be used to provide the hessian and Jacobians for each of the blocks. More details on this are given below.

4.2.4 `NLP` and Derived Classes

The `InteriorPointSolver` interacts with the problem representation through the `NLP` base class. This class has a number of pure virtual functions that must be implemented in the derived classes to provide information and evaluation of the model. The public methods of the NLP class are shown below.

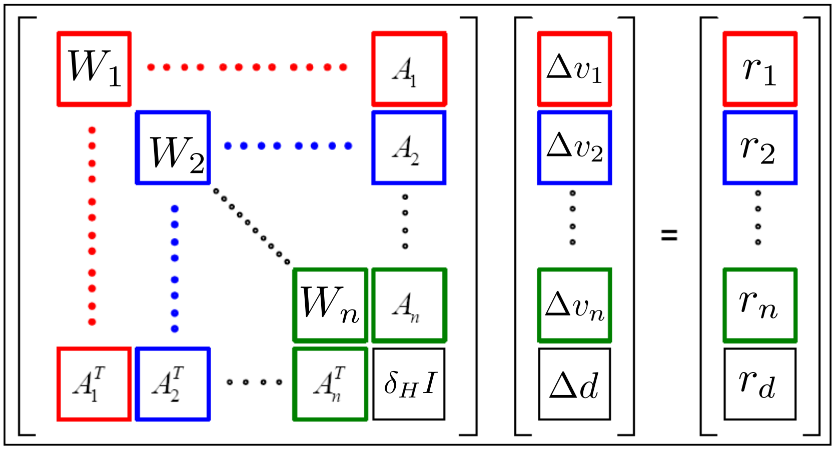


Figure 4.9: Representation of the permuted KKT system with blocks colored according to process ownership

Public Member Functions

- NLP ()

  Standard constructor

- virtual ∼NLP ()

  Standard destructor

- virtual int Nx () const =0

  Return the number of variables in the problem

- virtual int Nc () const =0

  Return the number of constraints in the problem

- virtual Vector ∗ create_new_x_vector () const =0

  Return a new instance of the variable vector as a `Vector` base class. This method is necessary since the `InteriorPointSolver` does not know about specific derived classes. It will instead call this method when it needs a new x

vector, and the method will create the derived class, and return a pointer to the base class.

- virtual Vector * create_new_c_vector () const =0

  Return a new instance of the constraint vector as a `Vector` base class. This method is necessary since the `InteriorPointSolver` does not know about specific derived classes. It will instead call this method when it needs a new c vector, and the method will create the derived class, and return a pointer to the base class.

- virtual Matrix * create_new_jc_matrix () const =0

  Return a new instance of the Jacobian matrix as a `Matrix` base class. This method is necessary since the `InteriorPointSolver` does not know about specific derived classes. It will instead call this method when it needs a new Jacobian instance, and the method will create the derived class, and return a pointer to the base class.

- virtual Matrix * create_new_hes_lag_matrix () const =0

  Return a new instance of the Hessian matrix as a `Matrix` base class. This method is necessary since the `InteriorPointSolver` does not know about specific derived classes. It will instead call this method when it needs a new Hessian instance, and the method will create the derived class, and return a pointer to the base class.

- virtual void get_x_l (Vector &xl) const =0

  Return the lower bounds on x

- virtual void get_x_u (Vector &xu) const =0

  Return the upper bounds on x

- virtual void get_x_init (Vector &x_init) const =0

  Return the initial values for x

- virtual void eval_obj (const Vector &x, double &f)=0

  Return the value of the objective function at the point x

- virtual void eval_equal_con (const Vector &x, Vector &c)=0

  Return the residuals of the equality constraints at the point x

- virtual void eval_all (const Vector &x, const Vector &lam_c, double &f, Vector &c, Vector &deriv_f, Matrix &jac_c, Matrix &hes_lag)=0

  Return the value of the all model quantities at the point x

- virtual void report_solution (const Vector &x, const Vector &lam_c)

  Once the algorithm has solved, it will call this method to allow the `NLP` to report the solution appropriately.

There are a number of specific implementations of the `NLP` class, as shown in the class diagram given in Figure 4.10.

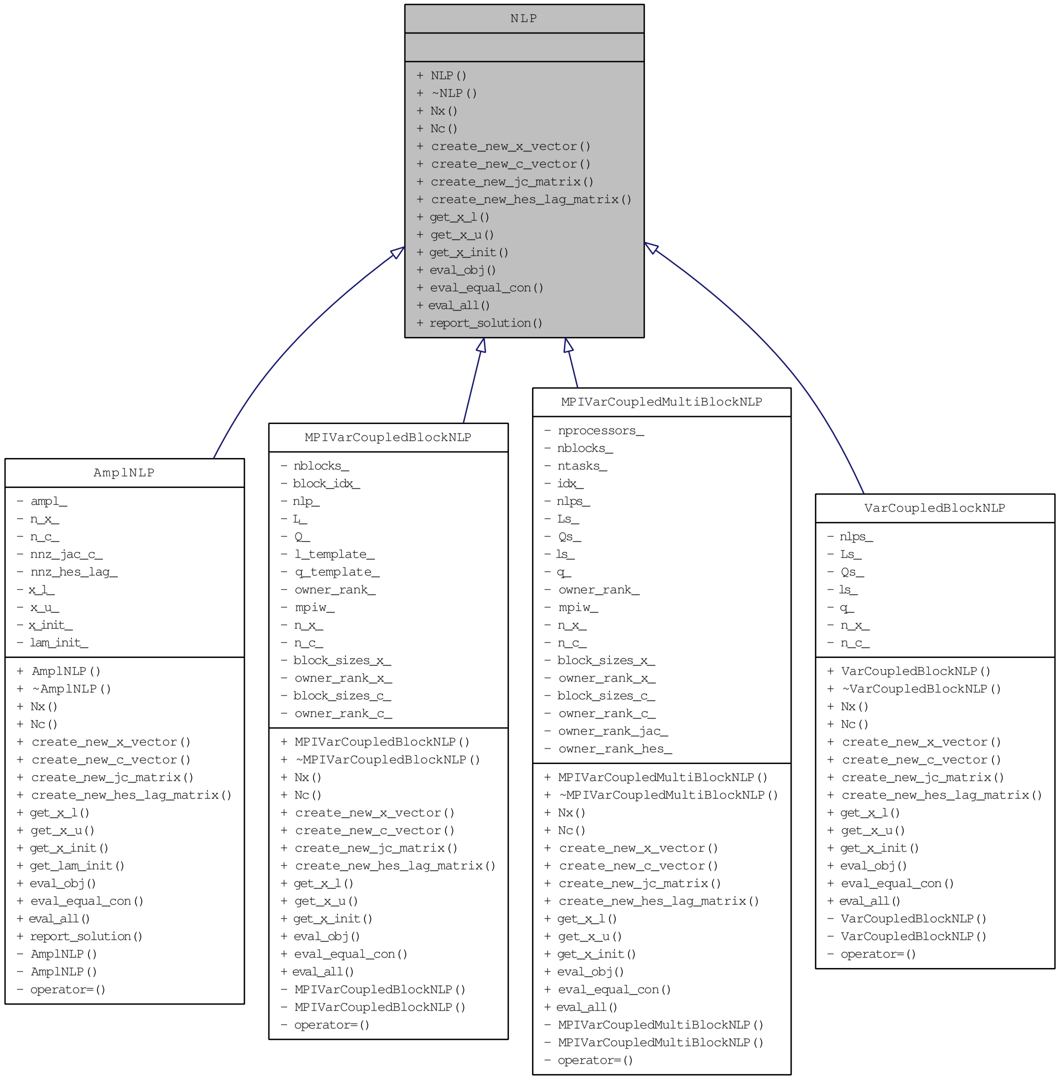


Figure 4.10: Class diagram for NLP

To understand how the derived NLP classes are used, we first consider again the structured nonlinear programming problem,

$$
\begin{aligned}
\min_{x_l, d} \quad & \sum_{l \in \mathcal{N}} f_l\left(x_l\right) & \\
\text{s.t.} \quad & c_l\left(x_l\right) = 0 & \forall\ \ l \in \mathcal{N} \\
& \underline{x}_l \leq x_l \leq \bar{x}_l & \forall\ \ l \in \mathcal{N} \\
& P_l x_l - P_l^d d = 0 & \forall\ \ l \in \mathcal{N}.
\end{aligned}
$$

As indicated in Section 3 the structured NLP, can be easily reconstructed from separate instances of a nonlinear programming problem for each block and some description of the indices of the dummy coupling variables within each block. The objective function is simply a sum over the different blocks, and the coupling can be easily described by defining the permutation matrices. Therefore, in this software implementation, the decomposition algorithms are actually given a list of NLP representations. They can then build the full problem (in serial, or distributed in parallel) to send to the `InteriorPointSolver`. In this fashion, the `VarCoupledBlockNLP` takes in a list of `NLP` objects (in the constructor, and implements the serial representation of structured problem shown above. Likewise, the parallel versions, `MPIVarCoupledBlockNLP`, and `MPIVarCoupledMultiBlockNLP` take in a list of `NLP` objects (where NULL values are present if a particular block is not owned by this process), and aggregate these objects into a parallel distributed representation of the structured problem. This is a common composite software design pattern.

Our implementation also includes `AmplNLP`, an interface to the Ampl Solver Library (ASL) (Fourer et al., 1993), that allows solution of problems formulated in `.nl` format. This is the format used in the common modeling environment, AMPL, and other modeling languages, like Pyomo. The decomposition implementations make use of a separate .nl file for each block in the problem with separate instances of the AMPL Solver Library for each block, allowing parallel evaluation of the NLP quantities. As shown in Figure 4.11, each block has its own NLP object, which is actually an instance of an `AmplNLP`. Through the AMPL solver library, each NLP object provides the evaluation of objectives, constraints, gradients, hessians, etc, and the higher-level NLP performs any necessary aggregation.

In order to generate nl files, we can use g prefixing in AMPL or AMPL Problem Writer Plugin in Pyomo (Hart et al., 2012). There are several advantages in using Pyomo. First, since it is a modeling language built upon python, it supports the definition and solution of optimization applications using the rich Python scripting environment. It is portable, can be used on most platforms. Moreover, it can produce nl files for NLP problems, thus can be interfaced with any AMPL solvers. Finally, it has flexible, extensible modeling and optimization capabilities. It allows custom extensions for parallel construction and solution. It also allows the modelers to leverage the high-level programming constructs for specific problem classes.

Figure 4.12 shows how we use Pyomo to build our custom parallel formulation. First, based on the mathematical algebra of problem formulation, we build the related Pyomo model, and generate the corresponding nl file for each block (process) using the AMPL Problem Writer Plugin. Then for each nl file, we generate one AMPL Solver Library (ASL) Object. Each ASL Object belongs to one process.

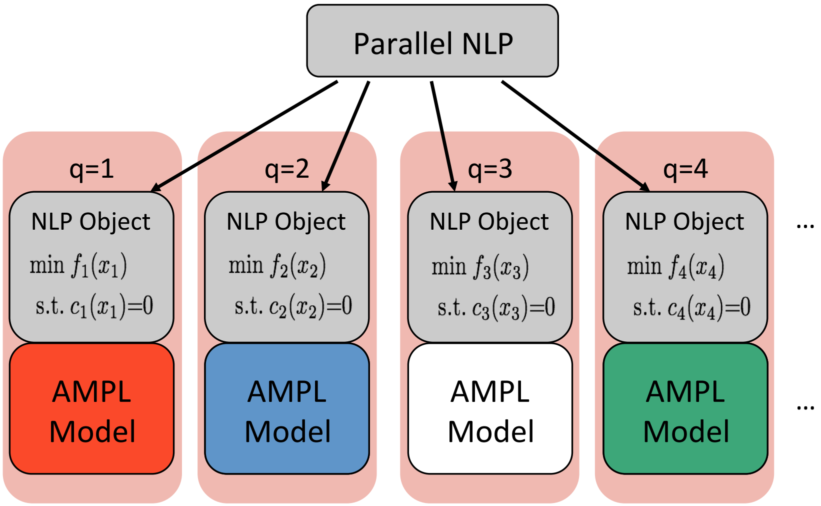


Figure 4.11: Parallel implementation with AMPL Solver Library (ASL) interfaced NLP objects

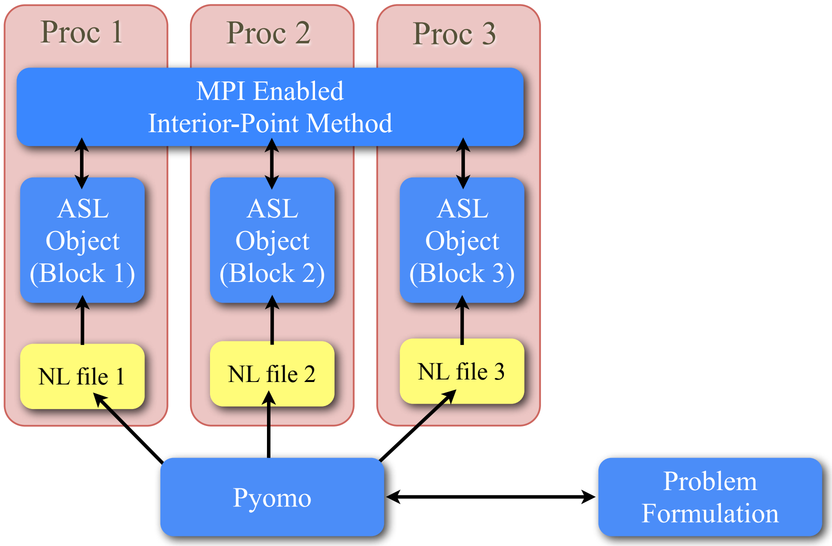


Figure 4.12: Custom parallel implementation with Pyomo

5. STOCHASTIC OPTIMAL POWER FLOW WITH EXPLICIT SCHUR-COMPLEMENT APPROACH*

The parallel explicit Schur-complement approach based on nonlinear interior-point methods is described in the previous few sections. This approach has been demonstrated to have excellent scalability properties on several structured, large-scale nonlinear programming problems from a variety of application areas (Word et al., 2014; Laird and Biegler, 2008; Zavala et al., 2008; Zhu et al., 2011). In this section, we test the performance of our implementation of the serial and parallel explicit Schur-complement approach (ESC-S, and ESC-P) against the serial full-space approach (FS-S) on an important problem in the area of optimal power management.

5.1 Contingency-constrained Alternating Current Optimal Power Flow (ACOPF)

In the traditional alternating current optimal power flow (ACOPF) problem we seek to find the optimal generator setpoints that minimize operational costs while satisfying load demands across an electrical transmission network. The ACOPF is an important problem that has been studied for five decades, but solution of this optimization problem is still challenging because of the large size of the system and the nonlinearities of the model. The Federal Energy Regulatory Commission (FERC) website lists several papers (Cain et al., 2012; O'Neill et al., 2012b,a; Castillo and O'Neill, 2013b,a; Schecter and O'Neill, 2013; Lipka et al., 2013; Pirnia et al., 2013; Campaigne et al., 2013) about the history, formulation, approximation, and solution

*Part of this section is reprinted with permission from "Parallel solution of nonlinear contingency-constrained network problems" by Kang, J., Siirola, J.D., Watson J. and Laird, C.D., 2014. In: Proceedings of the 8th International Conference on Foundations of Computer-Aided Process Design FOCAPD 2014, July 13-17, 2014, Cle Elum, Washington, USA, Copyright 2014 Elsevier B.V.

of the ACOPF problem. In their studies, they conclude that the rectangular IV formulation performs better than the polar formulation for larger problems.

The electrical grid is a critical infrastructure, and in addition to reducing operational costs, one would like to guarantee that the system is resilient to failure. In this section, we consider an extension to the traditional ACOPF problem, called the contingency-constrained ACOPF problem. Using a rectangular IV formulation to model AC power flow in the transmission network, we construct a nonlinear, multi-scenario optimization formulation that minimizes the operating cost for the nominal case while including a large number of contingency scenarios, where each scenario considers failure of an individual transmission element. We solve the corrective form of the problem, with terms added to the objective function to penalize deviations in the control variables between the nominal case and the contingency case. Given the number of potential failures in the network, these problems can become very large; yet we need to solve them efficiently to rapidly determine new optimal operating conditions over changing network demands.

For a realistic power network, with numerous contingencies considered, the overall problem size will increase dramatically, quickly exhausting the capabilities of a single workstation. Fortunately, the structure of these multi-scenario problems can be exploited to allow solution in parallel (Borges and Alves, 2007; Phan and Kalagnanam, 2012). This problem is nonlinear (and non-convex), and interior-point methods provide a powerful tool for local solution of these formulations (Capitanescu et al., 2006; Phan and Kalagnanam, 2012). The dominant computational expense in an interior-point method is the solution a linear system at each iteration arising from a modified Newtons method, and a number of researchers have investigated structure-exploiting approaches to allow parallel solution of this structured linear system (Qiu et al., 2005; Lubin et al., 2011; Chiang et al., 2014; Kang et al., 2014).

In this section, we show that fast solution of the contingency-constrained ACOPF problem is possible with our parallel implementation of the explicit Schur-complement approach. The remainder of this section is organized as follows. Section 5.2 describes the contingency-constrained ACOPF formulation. Numerical solution and timing results are shown and discussed in Section 5.3. Finally, we present a numerical comparison of our approach with an external partitioning approach, the progressive hedging algorithm (Rockafellar and Wets, 1991) available in PySP as part of the Pyomo modeling package.

5.2 Problem Formulation

When modelling the transmission network, we consider a rectangular IV formulation. This model, shown below in equations (1a-1q), is based on the traditional model for transmission lines as derived in the Matpower Users Manual (Zimmerman and Murillo-S'anchez, 2011). All sets, parameters, and variables used in the model are given in Table 1. The model includes relationships between voltages and current as defined by the transmission network, bus current balances, constraints on required power for all loads, and bounds that constrain operation within reasonable limits.

$$
\text{s.t.}\quad \left.\begin{aligned}
\begin{bmatrix} i^l_{fr} \\ i^l_{fj} \\ i^l_{tr} \\ i^l_{tj} \end{bmatrix} &= Y_{br} \begin{bmatrix} v^l_{fr} \\ v^l_{fj} \\ v^l_{tr} \\ v^l_{tj} \end{bmatrix} \\
\begin{bmatrix} i^l_{Sr} \\ i^l_{Sj} \end{bmatrix} &= Y_{sh} \begin{bmatrix} v^b_{r} \\ v^b_{j} \end{bmatrix} \\
P^l_t &= (v^{b(t)}_r \cdot i^l_{tr} + v^{b(t)}_j \cdot i^l_{tj}) \\
Q^l_t &= (v^{b(t)}_j \cdot i^l_{tr} - v^{b(t)}_r \cdot i^l_{tj}) \\
P^l_f &= (v^{b(f)}_r \cdot i^l_{fr} + v^{b(f)}_j \cdot i^l_{fj}) \\
Q^l_f &= (v^{b(f)}_j \cdot i^l_{fr} - v^{b(f)}_r \cdot i^l_{fj}) \\
S^l_t &= (P^l_t)^2 + (Q^l_t)^2 \\
S^l_f &= (P^l_f)^2 + (Q^l_f)^2 \\
0 &= \sum_{l \in \mathcal{B}^b_{in}} i^l_{tr} + \sum_{l \in \mathcal{B}^b_{out}} i^l_{fr} + i^b_{Sr} + i^d_{Lr} - \sum_{g \in \mathcal{G}^b} i^g_{Gr} \\
0 &= \sum_{l \in \mathcal{B}^b_{in}} i^l_{tj} + \sum_{l \in \mathcal{B}^b_{out}} i^l_{fj} + i^b_{Sj} + i^d_{Lj} - \sum_{g \in \mathcal{G}^b} i^g_{Gj} \\
P^d_L &= (v^d_r \cdot i^d_{Lr} + v^d_j \cdot i^d_{Lj}) \\
Q^d_L &= (v^d_j \cdot i^d_{Lr} - v^d_r \cdot i^d_{Lj}) \\
P^g_G &= (v^g_r \cdot i^g_{Gr} + v^g_j \cdot i^g_{Gj}) \\
Q^g_G &= (v^g_j \cdot i^g_{Gr} - v^g_r \cdot i^g_{Gj}) \\
v^b_m &= (v^b_r)^2 + (v^b_j)^2 \\
v^{b(ref)}_j &= 0 \\
\text{bounds on } v^b_m&, P^g_G, Q^g_G, S^l_f, S^l_t
\end{aligned}\right\}
\begin{aligned} l &\in \mathcal{L} \\ b &\in \mathcal{B} \\ g &\in \mathcal{G} \\ d &\in \mathcal{D} \end{aligned}
\tag{5.1}
$$

Table 5.1: Set, Parameter and Variable Description

$\mathcal{L}$	set of all branches (transmission lines)
$\mathcal{B}$	set of all bus nodes
$\mathcal{G}$	set of all generators
$\mathcal{D}$	set of all buses that are loads (a subset of $\mathcal{B}$)
$\mathcal{B}^b_{in}$	all inlet branches to bus b
$\mathcal{B}^b_{out}$	all outlet branches from bus b
$\mathcal{G}^b$	all generators at bus b
v^b_r, v^b_j	real and complex components of the voltages (at each bus b)
v^b_m	square of voltage magnitude (at each bus b)
P^d_L, Q^d_L	P and Q for each load d
P^g_G, Q^g_G	P and Q for each generator g
P^l_f, Q^l_f	P and Q at the $from$ end of each branch (transmission line l)
P^l_t, Q^l_t	P and Q at the *to* end of each branch (transmission line l)
S^l_f	S at the $from$ end of each branch (transmission line l)
S^l_t	S at the *to* end of each branch (transmission line l)
i^l_{fr}, i^l_{fj}	real and complex components of the current at the $from$ end of each branch l
i^l_{tr}, i^l_{tj}	real and complex components of the current at the *to* end of each branch l
i^d_{Lr}, i^d_{Lj}	real and complex components of the current for each load d
i^b_{Sr}, i^b_{Sj}	real and complex components of the current for the shunts
i^g_{Gr}, i^g_{Gj}	real and complex components of the current for generators

When formulating the multi-scenario contingency-constrained model, we repeat these equations for each contingency case, except we assume a single line failure (modifying the corresponding entries in the transmission matrix) for each scenario. The full multi-scenario optimization formulation is shown in equations (5.2) below, where x_0 and x_c represent the state variables for normal operation (i.e., no failure occurs) and the contingency cases, respectively. Vectors u_0 and u_c represent the control variables for normal operation and the contingency cases (active generator power in our studies).

$$\min_{x_0,u_0,x_c,u_c} \quad f(u_0) + \rho \sum_{c\in\mathcal{C}} f_c\left(u_c, u_0\right) \tag{5.2a}$$

$$\text{s.t.} \quad g_0\left(x_0, u_0\right) = 0 \tag{5.2b}$$

$$g_c\left(x_c, u_c\right) = 0 \qquad \forall\ c \in \mathcal{C} \tag{5.2c}$$

$$\underline{x}_0 \leq x_0 \leq \bar{x}_0,\ \underline{u}_0 \leq u_0 \leq \bar{u}_0 \tag{5.2d}$$

$$\underline{x}_c \leq x_c \leq \bar{x}_c,\ \underline{u}_c \leq u_c \leq \bar{u}_c \qquad \forall\ c \in \mathcal{C} \tag{5.2e}$$

Equations (5.2b) represent the complete network model corresponding to normal operation, while equations (5.2c) represent the network models for each of the contingency. The function $f(u_0)$ is a polynomial describing the generator operating cost. The penalty function $f_c\left(u_c, u_0\right)$ is the sum of the square of the 2-norm of the deviation between each vector u_c and u_0. However, other measures could be considered instead, such as ramp rate constraints.

5.3 Parallel Timing Results

Problem (5.2) represents a large-scale nonlinear, non-convex optimization problem where each scenario is independent except for the coupling in the penalty term between the normal case and the contingency cases. This multi-scenario problem is formulated using Pyomo (Hart et al., 2012), a Python-based, mathematical programming language. Pyomo allows formulation of the problem in parallel and produces *nl* files that can be processed by our implementation to allow model evaluation and solution in parallel.

For numerical timing, we consider the problem `case118` distributed with Matpower 4.1. This test problem has 118 buses, 54 active generators, and 186 branches. We first test our model by solving the single-scenario ACOPF problem with no contingencies using our interior-point solver and comparing this with the optimal results produced with Matpower. Both codes can solve the single-scenario ACOPF problem quickly (less than a second on a 2.3 GHz Intel Core i5 MacBook Pro), obtaining the same objective function value (129660.69 \$/hr) and generator setpoints.

Next, we report timing on the multi-scenario problem with 128 scenarios in total. We consider the normal operating scenario and 127 contingencies. The extensive form of this problem contains approximately $400,000$ variables and $385,000$ constraints. All timing results are wall-clock times obtained from the Red Mesa supercomputing cluster at Sandia National Laboratories in Albuquerque, NM. This cluster is made up of computing nodes each with two, 2.93 GHz quad-core, Nehalem X5570 processors (giving 8 computing cores per node). Each node has 12 GB of DDR3 RAM. To compute parallel speedup, we first solve the problem with two serial options in our interior-point algorithm. The FS-S algorithm solves the full-space KKT system using a direct linear solver, while the ESC-S algorithm solves the KKT system with the explicit Schur-complement decomposition approach. FS-S solves the problem in 199 seconds and, as expected, the ESC-S algorithm is slower, solving the problem in 412 seconds. Next, we compare these timing results with the ESC-P algorithm, the parallel explicit Schur-complement approach. Table (5.2) lists the timing results for this problem as we increase the number of processors.

The total wall-clock time for solving this problem can be decreased dramatically to less than 5 seconds with 128 processors. This represents an overall speedup of approximately 90 times when compared to the ESC-S approach, and over 40 times when compared with the FS-S approach.

Table 5.2: Strong Scaling Results for 128 Scenarios (127 Contingencies)

# processors	ESC-P Time(s)	Speedup (based on ESC-S)	Speedup (based on FS-S)
1	412.18	1.00	0.48
2	215.35	1.91	0.92
4	110.21	3.74	1.80
8	60.99	6.76	3.26
16	31.72	13.00	6.26
32	16.05	25.68	12.37
64	8.63	47.75	23.01
128	4.55	90.54	43.63

5.4 Comparison of Explicit Schur-Complement Approach with Progressive Hedging Algorithm in Pyomo

In Sections 1 and 3 we discussed two broad classes of algorithms for parallel decomposition in nonlinear programming: approaches that perform external partitioning and approaches that perform internal partitioning. The class of algorithms developed as part of this dissertation all employ internal partitioning. Recall that while the external partitioning approach is less intrusive and easier to implement, convergence rates and robustness are less favorable. In fact, many of these strategies do not have convergence guarantees on general nonlinear problems. On the other hand, the internal approach is more intrusive and harder to implement, but retains favorable convergence properties of the host algorithm employed.

In this section, we compare parallel solution times of an external partitioning approach, the progressive hedging (PH) algorithm, with the explicit Schur-complement approach on a variation of the contingency-constrained ACOPF problem described earlier in this section. The PH approach is selected since Pyomo already includes an interface between PySP (a stochastic programming extension) and the PH algorithm.

The progressive hedging algorithm is a special case of the *alternating direction method of multipliers* (ADMM). Here, the basic idea behind progressive hedging is described in the context of two-stage stochastic programming. The PH approach allows for decoupling of scenarios by relaxing the non-anticipativity constraints to produce independent subproblems. This approach then penalizes the deviation of the first-stage variables with respect to estimates of the anticipative value for the first-stage variables (selected as a weighted average between all scenarios from the last iteration). This approach can be made parallel by solving the independent subproblems in parallel, while an outer loop serves to converge the first-stage variables. Further details of this approach can be found in Rockafellar and Wets (1991); Feng et al. (2015) and Gade et al. (2014).

5.4.1 Modified CCOPF Model[†]

This section describes the two-stage stochastic programming formulation for the contingency-constrained ACOPF (CCOPF) that was modified to suite the PySP framework. The first stage contains an optimal AC power flow formulation that considers the generator costs subject to meeting the desired load specifications and transmission constraints. The second stage considers a large number of potential contingencies, including the high-probability event that normal operation will continue, along with lower-probability events that consider breakage of one of the transmission elements. For each of these scenarios, we consider ramping constraints for the generator power between stage 1 and stage 2. We consider all $N-1$ contingencies except those that produced a standalone island with a single bus and a generator. This restriction removes very few potential contingencies from consideration.

[†] The model described in this section was taken from work done by Dr. Laird in conjunction with researchers at Sandia National Laboratories, and was not developed as part of this dissertation. All numerical timing runs were conducted as part of this dissertation.

The complete extensive form of the problem formulation is given below, along with a description of all the symbols in Table 5.3.

Table 5.3: List of Symbols for CCOPF Problem Formulation

Symbol	Description
$\mathcal{S}$	Set of all scenarios
$\mathcal{T}$	Set of all stages (two stages considered here)
$\mathcal{L}$	Set of all transmission lines (branches)
$\mathcal{B}$	Set of all buses
$\mathcal{G}$	Set of all generators
$\mathcal{H}$	Set of all buses with a shunt
$\mathcal{D}$	Set of all buses with a specified load
$\mathcal{I}_b$	Set of all transmission lines going into bus b
$\mathcal{O}_b$	Set of all transmission lines going out of bus b
$v^r_{b,t,s}$, $v^j_{b,t,s}$	Variables for real and complex voltage at bus b, stage t, and scenario s
$P^G_{g,t,s}$, $Q^G_{g,t,s}$	Variables for true (P) and reactive (Q) power for generator g, stage t, and scenario s
$P^L_{b,t,s}$, $Q^L_{b,t,s}$	Variables for true (P) and reactive (Q) power delivered as load at bus b, stage t, and scenario s
$\bar{P}^L_{b,t,s}$, $\bar{Q}^L_{b,t,s}$	Variables for the absolute value of the difference between desired and delivered true (P) and reactive (Q) power at bus b, stage t, and scenario s
$P^S_{b,t,s}$, $Q^S_{b,t,s}$	Variables for true (P) and reactive (Q) power through a shunt at bus b, stage t, and scenario s
$P^f_{l,t,s}$, $Q^f_{l,t,s}$	Variables for true (P) and reactive (Q) power on the ‘from’ side of line l for stage t and scenario s

Symbol	Description
$P^t_{l,t,s}$, $Q^t_{l,t,s}$	Variables for true (P) and reactive (Q) power on the 'to' side of line l for stage t and scenario s
$i^{fr}_{l,t,s}$, $i^{fj}_{l,t,s}$	Variables for real and complex current on the 'from' side of line l for stage t and scenario s
$i^{tr}_{l,t,s}$, $i^{tj}_{l,t,s}$	Variables for real and complex current on the 'to' side of line l for stage t and scenario s
$P^{\star L}_b$, $Q^{\star L}_b$	Parameters for real (P) and reactive (Q) power desired as load at bus b
$P^{\star G}_{g,t,s}$, $Q^{\star G}_{g,t,s}$	Parameters for typical true (P) and reactive (Q) power for generator g, stage t, and scenario s
p_s	Parameter values for the probability of scenario s
$C^G_g(\cdot)$	Cost function for generator g dependent on true and reactive power.
C^L	Parameter for the value of lost load (incremental cost of not meeting demand)
$Y_{l,t,s}$	Parameter matrix describing IV relationships for branch l, stage t, and scenario s
Y^S_b	Parameter for shunt equation for bus b
S^U	Parameter for the upper bound on apparent power
P^{GL}_g, P^{GU}_g	Parameters for the lower and upper bound on true (P) power for generator g
Q^{GL}_g, Q^{GU}_g	Parameters for the lower and upper bound on reactive (Q) power for generator g
P^{GR}_g	Parameters specifying the maximum ramp rate between stage 1 and 2 for generator g

Symbol	Description
V_b^L, V_b^U	Parameter for the lower and upper bound on voltage magnitude for bus b
rb	Index of reference bus
$bf(l)$, $bt(l)$	Function for the index of bus connected to the 'from' and 'to' side of branch l

The stochastic programming formulation seeks to minimize the expected value of operating cost across all the scenarios, indicated by $\mathcal{S}$. Here, scenario index 0 refers to the scenario with continued normal operation, and the remaining indices refer to contingencies. We consider a two-stage problem. Therefore the set of stages $\mathcal{T}$ contains indices for the first-stage (1) and second-stage (2) only. Each stage includes a full ACOPF formulation, therefore, each scenario contains two ACOPF formulations, one for stage 1, and one for stage 2. For each stage and each scenario, we consider a full nonlinear AC model for the transmission, and for large networks, the extensive form is an extremely large, nonlinear stochastic programming problem that is not tractable with standard desktop computing resources.

$$\begin{aligned}\min \sum_{s\in S} p_s \sum_{t\in T} \Bigg[\sum_{g\in\mathcal{G}} C_g^G(P_{g,t,s}^G, Q_{g,t,s}^G) + \rho_1 \sum_{g\in\mathcal{G}} \left[\left(P_{g,t,s}^G - P_{g,t,s}^{\star G}\right)^2 + \left(Q_{g,t,s}^G - Q_{g,t,s}^{\star G}\right)^2 \right] \\ +\rho_2 \sum_{b\in\mathcal{D}} \left[\left(P_{b,t,s}^L - P_b^{\star L}\right)^2 + \left(Q_{b,t,s}^L - Q_b^{\star L}\right)^2 \right] \Bigg]\end{aligned} \tag{5.3}$$

$$\text{s.t.} \quad \begin{bmatrix} i^{fr}_{l,t,s} \\ i^{fj}_{l,t,s} \\ i^{tr}_{l,t,s} \\ i^{tj}_{l,t,s} \end{bmatrix} = Y_{l,t,s} \begin{bmatrix} v^{r}_{bf(l),t,s} \\ v^{j}_{bf(l),t,s} \\ v^{r}_{bt(l),t,s} \\ v^{j}_{bt(l),t,s} \end{bmatrix} \qquad \forall\, l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.4}$$

$$P^{S}_{b,t,s} = Y^{S}_{b} \left[(v^{r}_{b,t,s})^2 + (v^{j}_{b,t,s})^2 \right] \qquad \forall\, b \in \mathcal{H},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.5}$$

$$Q^{S}_{b,t,s} = -Y^{S}_{b} \left[(v^{r}_{b,t,s})^2 + (v^{j}_{b,t,s})^2 \right] \qquad \forall\, b \in \mathcal{H},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.6}$$

$$0 = \sum_{l \in \mathcal{I}_b} P^{t}_{l,t,s} + \sum_{l \in \mathcal{O}_b} P^{f}_{l,t,s} + P^{S}_{b,t,s} + P^{L}_{b,t,s} - P^{G}_{b,t,s} \qquad \forall\, b \in \mathcal{B},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.7}$$

$$0 = \sum_{l \in \mathcal{I}_b} Q^{t}_{l,t,s} + \sum_{l \in \mathcal{O}_b} Q^{f}_{l,t,s} + Q^{S}_{b,t,s} + Q^{L}_{b,t,s} - Q^{G}_{b,t,s} \qquad \forall\, b \in \mathcal{B},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.8}$$

$$P^{f}_{l,t,s} = v^{r}_{bf(l),t,s} \cdot i^{fr}_{l,t,s} + v^{j}_{bf(l),t,s} \cdot i^{fj}_{l,t,s} \qquad \forall\, l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.9}$$

$$Q^{f}_{l,t,s} = v^{j}_{bf(l),t,s} \cdot i^{fr}_{l,t,s} - v^{r}_{bf(l),t,s} \cdot i^{fj}_{l,t,s} \qquad \forall\, l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.10}$$

$$P^{t}_{l,t,s} = v^{r}_{bt(l),t,s} \cdot i^{tr}_{l,t,s} + v^{j}_{bt(l),t,s} \cdot i^{tj}_{l,t,s} \qquad \forall\, l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.11}$$

$$Q^{t}_{l,t,s} = v^{j}_{bt(l),t,s} \cdot i^{tr}_{l,t,s} - v^{r}_{bt(l),t,s} \cdot i^{tj}_{l,t,s} \qquad \forall\, l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.12}$$

$$v^{j}_{rb,t,s} = 0 \qquad \forall\, t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.13}$$

$$P^{L}_{b,1,s} = P^{\star L}_{b} \qquad \forall\, b \in \mathcal{D},\ s \in \mathcal{S} \tag{5.14}$$

$$Q^{L}_{b,1,s} = Q^{\star L}_{b} \qquad \forall\, b \in \mathcal{D},\ s \in \mathcal{S} \tag{5.15}$$

$$(S^{U})^2 \geq (P^{f}_{l,t,s})^2 + (Q^{f}_{l,t,s})^2 \qquad \forall\, l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.16}$$

$$(S^{U})^2 \geq (P^{t}_{l,t,s})^2 + (Q^{t}_{l,t,s})^2 \qquad \forall\, l \in \mathcal{L},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.17}$$

$$(V^{L}_{b})^2 \leq (v^{r}_{b,t,s})^2 + (v^{j}_{b,t,s})^2 \leq (V^{U}_{b})^2 \qquad \forall\, b \in \mathcal{B},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.18}$$

$$P^{GL}_{g} \leq P^{G}_{g,t,s} \leq P^{GU}_{g} \qquad \forall\, g \in \mathcal{G},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.19}$$

$$Q^{GL}_{g} \leq Q^{G}_{g,t,s} \leq Q^{GU}_{g} \qquad \forall\, g \in \mathcal{G},\ t \in \mathcal{T},\ s \in \mathcal{S} \tag{5.20}$$

$$-P^{GR}_{g} \leq P^{G}_{g,1,s} - P^{G}_{g,2,s} \leq P^{GR}_{g} \qquad \forall\, g \in \mathcal{G},\ s \in \mathcal{S} \tag{5.21}$$

$$P^G_{g,1,0} = P^G_{g,2,0} \qquad \forall\, g \in \mathcal{G} \tag{5.22}$$

$$Q^G_{g,1,0} = Q^G_{g,2,0} \qquad \forall\, g \in \mathcal{G} \tag{5.23}$$

$$P^G_{g,1,0} = P^G_{g,1,s} \qquad \forall\, g \in \mathcal{G},\ s \in \mathcal{S}/\{0\} \tag{5.24}$$

$$Q^G_{g,1,0} = Q^G_{g,1,s} \qquad \forall\, g \in \mathcal{G},\ s \in \mathcal{S}/\{0\} \tag{5.25}$$

The objective function, shown in equation (5.3) is the expected value of the operating costs across all scenarios. The parameter p_s provides the probability of scenario s, $C^G_g(P^G_{g,t,s}, Q^G_{g,t,s})$ is the cost function for generator g, which is typically represented as a low-order polynomial function of the generator power output. This function also includes a penalty term for not meeting the desired load at the buses.

Equations 5.4-5.13 give the physical model for the grid for every stage and scenario. Equation (5.4) are the IV relationships for the transmission lines based on the traditional π transmission model. The functions $bf(l)$ and $bt(l)$ return the indices of the bus connected to the 'from' and 'to' side of branch l respectively. Equations (5.5,5.6) model the shunts at all buses in $\mathcal{H}$. Equations (5.7, 5.8) are power balances around the bus. These expressions replace the traditional current balances and, in our test problems, provide improved nonlinear convergence. In this formulation, power values are positive if they are providing power to the bus, while negative if they are drawing power from the bus, with the exception of the generator power which is always listed as positive (explaining the negative sign in the balance). Also, the shunt power, load power, and generator power terms do not exist on every bus, and are only included when the buses contain those individual elements. Equations (5.9-5.12) provide expressions for the power in the 'from' and 'to' sides of the transmission element as a function of the corresponding voltages and currents. These are necessary since there are bounds on the power allowed in individual transmission lines. Equation (5.13) sets the voltage angle for the reference bus to zero.

We want to ensure that all loads are satisfied during normal operation, and equations (5.14, 5.15) require that the loads in the first stage of each scenario match the specified loads from the case definition. Equations (5.16-5.18) provide bounds on power and voltage in the transmission elements, and equations (5.19, 5.20) provide operating limits for the generators.

Ramping limits are provided in the case description for the generators and enforced in equation (5.21). These constraints enforce a bound on the rate of change allowed in generator setpoint. Equations (5.22, 5.23) ensure that the optimal operating condition is the same in the first and second stage for the single scenario representing continued normal operation (scenario 0). Finally, equations (5.24, 5.25) are the non-anticipativity constraints linking the first-stage variables across all the scenarios.

5.4.2 Timing Results

The test problem `case118` is used again with 128 total scenarios (127 contingencies plus the continued nominal operation). Table 5.4 shows strong scaling results for both the explicit Schur-complement approach and the progressive hedging approach. The results of the explicit Schur-complement approach are not surprising, and match very closely with those of the previous section. The explicit Schur-complement approach outperforms the progressive hedging approach in terms of overall solution for all processor counts. Furthermore, the actual speedup value of the progressive hedging approach (based on the solution time for the full-space method) is quite poor, and lower than the explicit Schur-complement approach by about a factor of three. For this test problem, it should also be noted that the first-stage solution for all of the independent sub-problems was very close, and PH only required a few iterations for convergence. Given a more challenging problem, we expect significantly more iterations for PH and worse overall performance. More interesting is the speedup of the progressive hedging approach based on its own serial timing. This column of the table appears to show superlinear speedup, but this should be considered carefully. The overall solution time is worse than ESC-P, and the apparent superlinear speedup is actually due to inefficiencies in the subproblem distribution code when the number of processors is low (giving falsely high times for the serial code).

The results of this section show that the explicit Schur-complement approach is an effective technique for optimization of these challenging stochastic optimization problems on the power grid. Furthermore, these methods can be significantly faster than external partitioning approaches. Nevertheless, these problems have relatively few coupling variables (on the order of one hundred). For problems with significantly more coupling, alternative algorithms need to be developed for efficient solution. The numerical performance of the PCGSC approach on problems with more coupling variables is discussed in detail in the next section.

Table 5.4: Strong Scaling Comparison between ESC and PH

	Full-Space	Explicit Schur-Complement			Progressive Hedging		
# processors	Time(s)	Time(s)	Speedup (based on ESC-S)	Speedup (based on FS-S)	Time(s)	Speedup (based on PH-S)	Speedup (based on FS-S)
1.00	49.82	109.11	1.00	0.46	378.77	1.00	0.13
4.00	-	27.74	3.93	1.80	78.41	4.83	0.64
8.00	-	14.78	7.38	3.37	37.08	10.21	1.34
16.00	-	8.53	12.80	5.84	19.97	18.97	2.49

6. NUMERICAL PERFORMANCE OF THE IMPLICIT PCGSC ALGORITHM FOR PROBLEMS WITH SIGNIFICANT COUPLING*

In the previous section, the explicit Schur-complement (ESC) approach was used to achieve parallel speedup for a problem with relatively few coupling variables. As the number of coupling variables increase, however, the cost of forming and factorizing the Schur-complement becomes prohibitive. The implicit PCG Schur-complement (PCSSC) approach has been developed to overcome this bottleneck. In this section, we compare the numerical performance of the ESC and the PCGSC approaches on a test problem where the degree of coupling can be readily varied while keeping other problem characteristics constant.

The computational performance of these methods is compared using both strong scaling and weak scaling metrics. Strong scaling refers to the ability of a parallel implementation to provide speedup. Here, the problem size is kept constant while the number of processors used in parallel computation is increased. If the approach scales perfectly, the observed speedup will be the same as the number of processors used. Weak scaling, on the other hand, measures the ability of an implementation to tackle larger problems. Here, the problem size is increased proportionally with the number of processors. If the implementation scales perfectly, then the computation time will remain constant as the size of the problem and the number of processors is increased. Two case studies are considered. The first is a set of quadratic programming problems constructed to illustrate the effect of increased variable coupling, and weak scaling results are shown for this example. The second is an multi-scenario

*Part of this section is reprinted with permission from “An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition” by Kang, J., Cao, Y., Word, D., and Laird, C.D., 2014. In: Computers and Chemical Engineering 71 (2014), pp 563-573, Copyright 2014 by Elsevier.

dynamic optimization problem that solves for the operating profile of a distillation column where the model is governed by a set of differential equations (numerical results of this case study are presented in the next section). Both strong and weak scaling metrics are shown for this case study.

6.1 Scalable Test Problem (QP)

Initial performance analysis is based on the following quadratic programming problem based on minimum least-squares parameter estimation,

$$\begin{aligned}
\min_{y,q,\theta} \quad & \sum_{l \in \mathcal{N}} \|y_l - y_l^\star\|_2^2 & \\
\text{s.t.} \quad & y_l - Aq_l = 0 & \forall\ l \in \mathcal{N} \\
& P_l \begin{bmatrix} y_l \\ q_l \end{bmatrix} - P_l^d \theta = 0 & \forall\ l \in \mathcal{N} \\
& \underline{y}_l \leq\ y_l \leq \bar{y}_l & \forall\ l \in \mathcal{N} \\
& \underline{q}_l \leq\ q_l \leq \bar{q}_l & \forall\ l \in \mathcal{N}
\end{aligned} \tag{6.1}$$

where y_l and q_l are vectors of variables local to each block l. The vector θ is the set of variables that must be the same across all blocks (i.e. the common or coupling variables), and the number of coupling variables is n_d. The vectors $y_l^\star$ correspond to the known measurements for y_l in each block l. A is a constant matrix composed of a stack of tri-diagonal matrices, and the matrices P_l and P_l^d specify the relationship between the variables corresponding to each individual block and the common variables θ. In each block l, the dimension of y_l is 10000 and the dimension of q_l is 5000. Noisy measurement data, $y_l^\star$, are obtained by specifying θ and the remaining entries in each q_l, calculating y_l, and adding normally distributed noise with a standard deviation of 5%. We specify P_l such that it relates the first n_d entries in q_l to the common variables θ.

This problem formulation allows us to easily test the performance of these algorithms as n_d (the number of coupling variables) is increased, while keeping the remaining problem structure and size the same.

6.2 Parallel Timing Results for ESC and PCGSC Approaches

In this section, we present timing results where the number of blocks (or data sets) are set to 2, 4, 8, and 16, and the number of coupling variables are varied from 10 to 3200. Table 6.1 shows the wall clock time required for the solution of these problems. All timing was performed on the Red Mesa supercomputing cluster at Sandia National Laboratories in Albuquerque, NM. This cluster is made up of computing nodes each with two, 2.93 GHz quad-core, Nehalem X5570 processors (giving 8 computing cores per node). Each node has 12 GB of DDR3 RAM. All 5 techniques obtained the same solution on these problems, and all the problems required 8 interior-point iterations to converge.

The parallel timing results in Table 6.1 for ESC-P and PCGSC-P represent weak scaling results where the number of processors used is the same as the number of blocks in the problem. In this way, these results provide the expected performance as the size of the problem and the number of processors used are both increased. These weak scaling results are shown as a function of the number of coupling variables.

Comparing the FS-S method with the ESC-S method, one can see that the explicit Schur-complement technique is outperformed by the standard full-space approach when these problems are solved in serial. Nevertheless, when the number of coupling variables is small, the parallel version (ESC-P) still has benefit with reasonable speedup possible on larger problems with up to 100 coupling variables. Figure 6.1a shows the timing results for the explicit Schur-complement algorithm with only

10 coupling variables. Weak scaling results for ESC-P are good (very little increase in the solution time as the number of blocks and processors are increased), and reasonable speedup is obtained when using ESC-P. Figure 6.1b shows the timing results for the same algorithms with 3200 coupling variables. Here again, weak scaling results for the ESC-P algorithm are good with little increase in solution time as the number of blocks and processors are increased. However, the solution time itself is significantly longer than the standard FS-S approach.

Table 6.1 and Figure 6.1 clearly show the overall performance deterioration of the explicit Schur-complement approach compared with the full-space method as the number of coupling variables increases. This performance deterioration is directly attributable to the explicit formation and factorization of the Schur-complement itself. The computational effort required to form the Schur-complement is linear in the number of coupling variables, while the dense factorization of the Schur-complement is cubic in the number of coupling variables. Figure 6.2 shows the average time required to factor and solve the Schur-complement as a function of the number of coupling variables along with a cubic trendline. In addition, this figure shows the average time required to form the contribution of a single block to the Schur-complement along with a linear trendline. As expected, the time required to solve the dense Schur-complement increases dramatically as we increase the number of coupling variables. In particular, for 3200 coupling variables the total time spent in factorization of the Schur-complement (summed over all 8 iterations) is over 150 seconds, or approximately 82% of the entire time spent when solving in parallel. It is exactly this result that drove the development of the implicit Schur-complement approach.

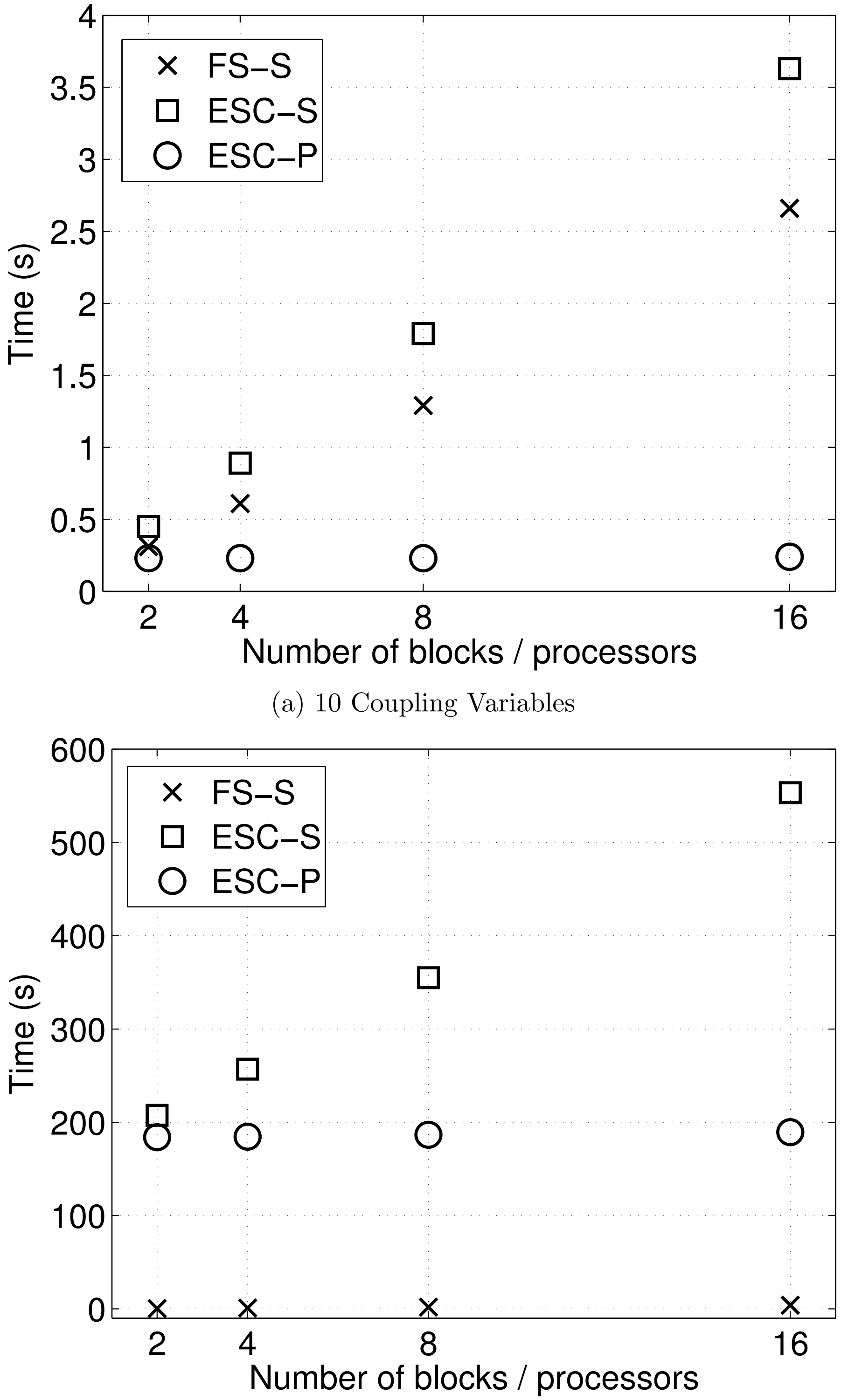
The PCGSC approach solves the Schur-complement system iteratively and removes the need for forming and factorizing the Schur-complement. Figures 6.3a and

6.3b show the weak-scaling results for the PCG Schur-complement method with 10 and 3200 coupling variables. As with the ESC-P algorithm, the weak-scaling results for the PCGSC-P approach are very good (there is little increase in execution time as we increase the number of blocks and processors). More importantly, however, these results do not display the significant performance degradation observed in the ESC-P approach for the case with 3200 coupling variables. In fact, the timing results in Table 6.1 show that the PCGSC-S and PCGSC-P approaches scale much more favorably than ESC-S and ESC-P as the number of coupling variables is increased. Furthermore, the PCGSC-P approach outperforms the FS-S approach in every case study with more than 2 blocks.

A direct comparison of the speedup and efficiency of the explicit Schur-complement and the PCG Schur-complement algorithms is shown in Figure 6.4. In these figures, we see that the efficiency of the explicit Schur-complement approach decreases much more dramatically than the PCG Schur-complement approach as the number of coupling variables increases. Compared with the explicit Schur-complement method, the computational cost of the PCG Schur-complement approach is almost two orders of magnitude better in the serial case, and over two orders of magnitude better in the parallel case for the problem with 3200 coupling variables. These results clearly demonstrate the benefit of this approach on problems with significant coupling.

Table 6.1: Timing Results for Quadratic Programming Problem

# Coupling Vars.	# Blocks	FS-S time(s)	ESC-S time(s)	ESC-P time(s)	PCGSC-S time(s)	PCGSC-P time(s)
10	2	0.31	0.45	0.23	0.41	0.21
	4	0.61	0.89	0.23	0.83	0.22
	8	1.29	1.79	0.23	1.74	0.22
	16	2.66	3.63	0.24	3.53	0.23
25	2	0.33	0.64	0.32	0.49	0.25
	4	0.61	1.27	0.32	0.99	0.25
	8	1.32	2.54	0.33	2.09	0.27
	16	2.67	5.20	0.34	4.34	0.27
50	2	0.31	0.95	0.48	0.59	0.30
	4	0.61	1.90	0.49	1.19	0.30
	8	1.30	3.81	0.50	2.52	0.32
	16	2.72	7.71	0.52	5.26	0.35
100	2	0.30	1.59	0.80	0.64	0.32
	4	0.62	3.20	0.81	1.29	0.33
	8	1.30	6.41	0.82	2.75	0.35
	16	2.73	12.78	0.84	5.66	0.36
200	2	0.32	2.93	1.47	0.65	0.33
	4	0.61	5.85	1.49	1.29	0.35
	8	1.32	11.59	1.50	2.74	0.34
	16	2.76	23.23	1.52	5.70	0.36
400	2	0.31	5.69	2.89	0.66	0.33
	4	0.63	11.11	2.93	1.29	0.33
	8	1.33	22.35	2.95	2.82	0.35
	16	2.85	44.47	3.00	5.89	0.37
800	2	0.32	11.90	6.54	0.68	0.35
	4	0.65	23.04	6.54	1.35	0.36
	8	1.40	44.51	6.60	2.92	0.37
	16	3.00	87.91	6.63	5.99	0.38
1600	2	0.34	34.04	22.68	0.70	0.38
	4	0.69	56.58	23.22	1.43	0.39
	8	1.55	102.47	23.42	3.15	0.40
	16	3.30	192.36	23.50	6.47	0.44
3200	2	0.37	207.47	184.01	0.80	0.45
	4	0.80	256.93	184.58	1.66	0.46
	8	1.78	354.99	186.67	3.69	0.47
	16	3.94	553.31	189.44	7.39	0.50



(a) 10 Coupling Variables

(b) 3200 Coupling Variables

Figure 6.1: Weak scaling results for the explicit Schur-complement method on QP test problems with different numbers of coupling variables

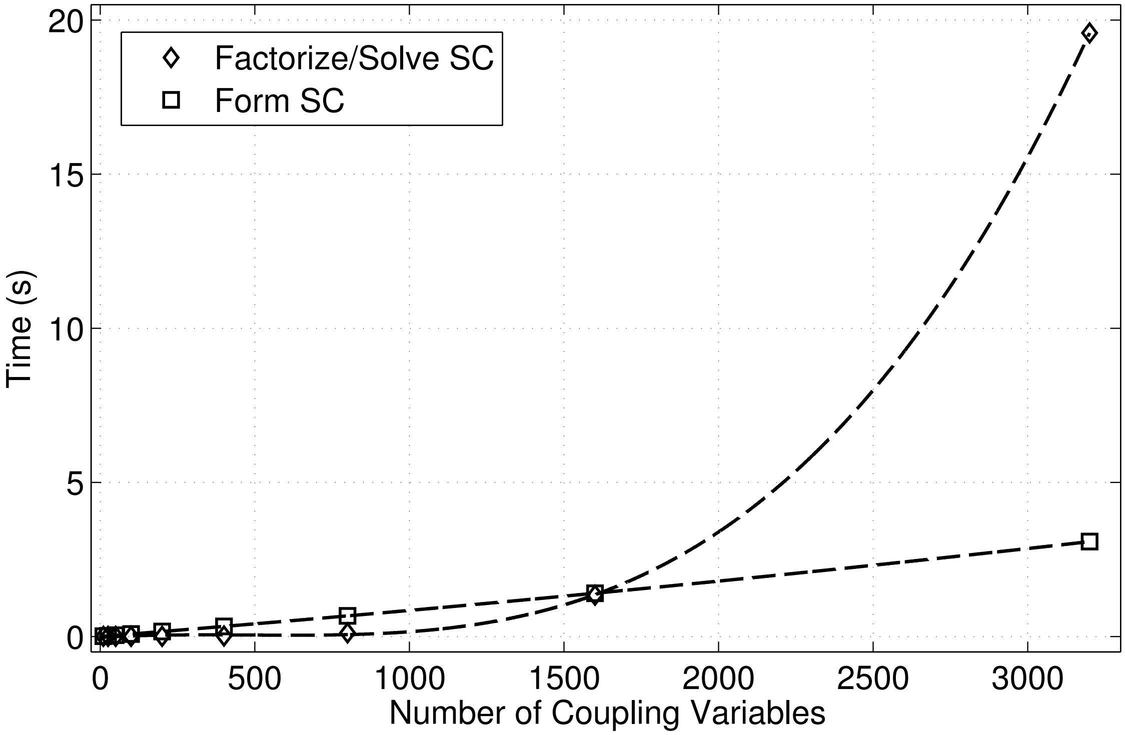
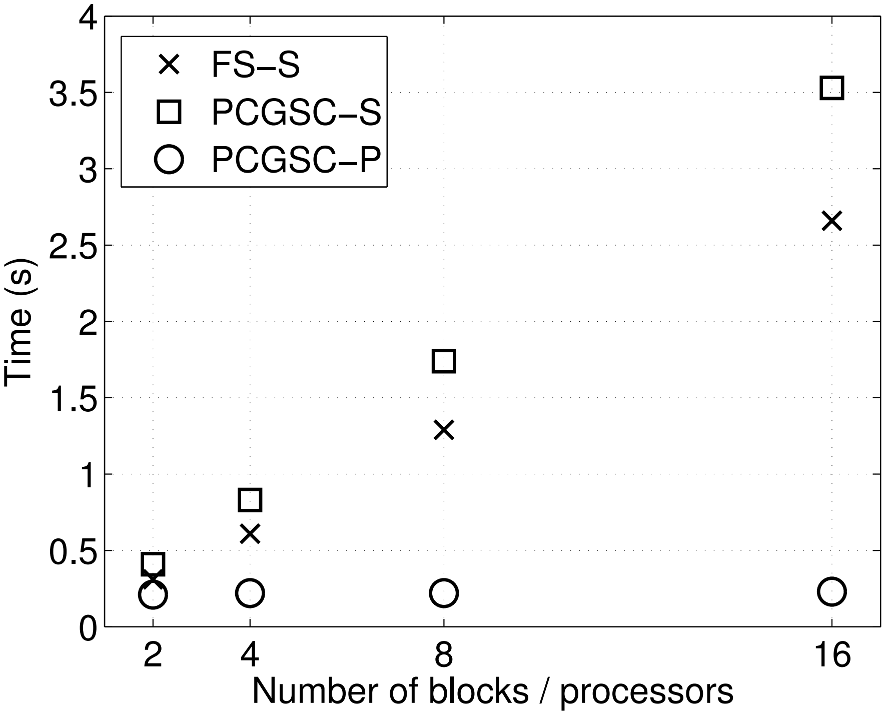
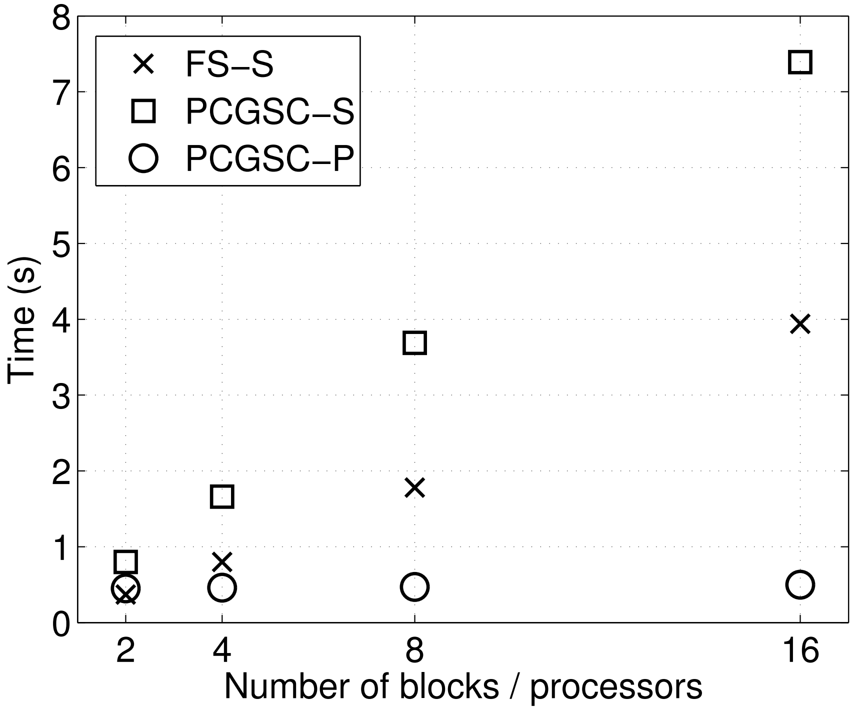


Figure 6.2: Time required to form and factor the Schur-complement as a function of the number of coupling variables

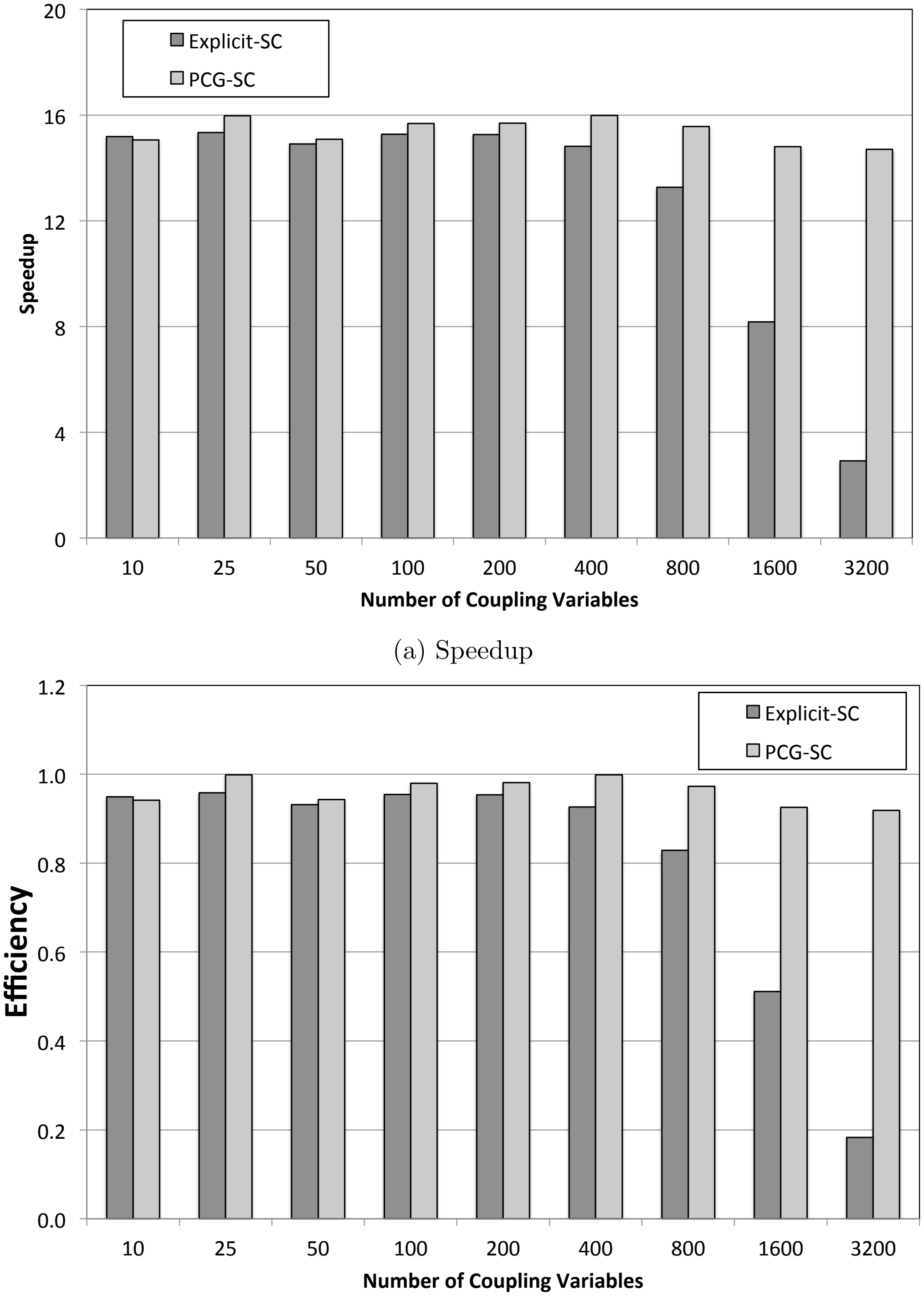


(a) 10 Coupling Variables



(b) 3200 Coupling Variables

Figure 6.3: Weak scaling results for the PCG Schur-complement method on QP test problems with different numbers of coupling variables



(a) Speedup

(b) Efficiency

Figure 6.4: Speedup comparison between explicit Schur-complement and PCG Schur-complement approaches

7. OPTIMAL OPERATION OF A DISTILLATION COLUMN UNDER UNCERTAINTY WITH IMPLICIT PCG SCHUR-COMPLEMENT APPROACH*

In the previous section, we showed that significant performance improvements were possible using the implicit PCG Schur-Complement (PCGSC) approach for problems with an increased number of coupling variables. This problem structure occurs in many different problem classes. In this section, we investigate the performance of the PCGSC algorithm on a realistic dynamic optimization problem. In dynamic optimization, there can be a large number of degrees of freedom associated with the control variables due to the need to discretize in time. In stochastic dynamic optimization, one wishes to perform a dynamic optimization while considering uncertainty in model parameters or system inputs. This class of problem fits the structure of nonlinear stochastic programming problem with significant coupling due to the large number of discretized first-stage variables. To further demonstrate the effectiveness of this approach, we also include timing results for a complex nonlinear optimization problem arising from dynamic optimization of a distillation process under uncertainty. In particular, we use this case study to evaluate the performance of the parallel algorithms ESC-P and PCGSC-P as the number of coupling variables is increased, and to demonstrate the weak and strong scaling properties of the PCGSC-P approach on a realistic nonlinear optimization problem.

*Part of this section is reprinted with permission from "An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition" by Kang, J., Cao, Y., Word, D., and Laird, C.D., 2014. In: Computers and Chemical Engineering 71 (2014), pp 563-573, Copyright 2014 by Elsevier.

7.1 Dynamic Distillation Column Model

The original model for the distillation unit is described in Hahn and Edgar (2002). This distillation column uses 30 trays to separate a binary mixture. Including the condenser and reboiler, it has 32 stages that are indexed from top to bottom, as shown in Figure 7.1. The feed stream is added at the 17^{th} stage. The complete model

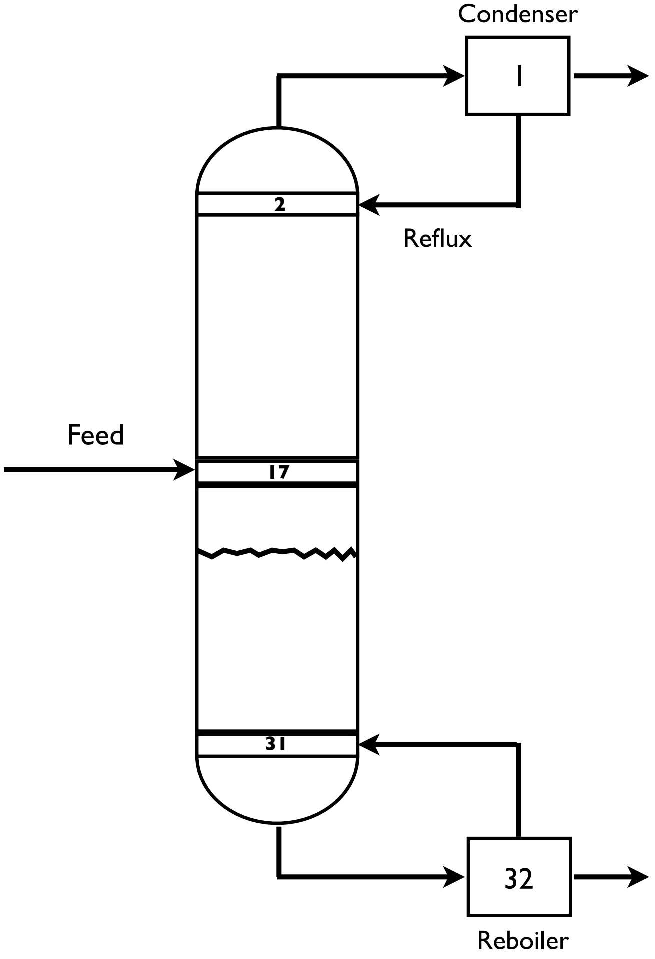


Figure 7.1: Flow diagram of the distillation column

contains 32 differential equations, 1 input, and 35 algebraic variables. The reflux ratio is defined as the control or input variable. Using the simultaneous approach (Biegler, 2010), we apply collocation on finite elements to discretize the differential equations, and write the complete time-discretized model as a large set of algebraic equations. The purity of the distillate, denoted as y_1, provides the measure of product quality. We assume that consumer demands change with time and seek an optimal trajectory for transition between the different operating points (specifications for y_1). We consider uncertainty in the mole fraction of the feed stream, and generate a multi-scenario problem formulation. The problem formulation (7.1) is shown below, where the parameter and variable descriptions are shown in Table 7.1. The symbols $\alpha_{A,B}$, F, A_{Cond}, A_{Tray} and $A_{Reboiler}$ are all constant physical parameters, whose values are set the same as in Column 1 of Table 4.2 in Horton (1987). X_k is the vector of all the variables in each block, u_c is the vector of coupling variables across all the blocks, and the matrices P_k and P_k^d specify the linking equations between the control variables in each block k and the common variables u_c. This notation is selected so that the matrices P_k and P_k^d match those used in Equation (1.3d).

$$\min \sum_{k=1}^{N}\sum_{l=1}^{N_k}\sum_{i=1}^{n_f}\sum_{j=1}^{n_c} h_{i,l,k}\Omega_{j,n_c}\left[\alpha\left(y_{1,i,j,l,k} - y_{1,i,j,l,k}^{set}\right)^2 + \rho\left(u_{i,j,k} - u_{i,j,k}^{set}\right)^2\right] \tag{7.1}$$

$$
\text{s.t.} \quad \left.\begin{aligned}
L1_{i,j,l,k} - u_{i,j,k} D_{l,k} &= 0 \\
V_{i,j,l,k} - L1_{i,j,l,k} - D_{l,k} &= 0 \\
L2_{i,j,l,k} - F - L1_{i,j,l,k} &= 0 \\
y_{n,i,j,l,k} - \frac{x_{n,i,j,l,k}\alpha_{A,B}}{1 + (\alpha_{A,B} - 1) x_{n,i,j,l,k}} &= 0, \\
n &= 1, ..., 32 \\
z_{1,i,j,l,k} - \frac{1}{A_{cond}} V_{i,j,l,k} (y_{2,i,j,l,k} - x_{1,i,j,l,k}) &= 0 \\
z_{n,i,j,l,k} - \frac{1}{A_{Tray}} [L1_{i,j,l,k} (x_{n-1,i,j,l,k} - x_{n,i,j,l,k}) - V_{i,j,l,k} (y_{n,i,j,l,k} - y_{n+1,i,j,l,k})] &= 0, \\
n &= 2, ..., 16 \\
z_{17,i,j,l,k} - \frac{1}{A_{Tray}} [F x_{Feed,l,k} + L1_{i,j,l,k} x_{16,i,j,l,k} & \\
-L2_{i,j,l,k} x_{17,i,j,l,k} - V_{i,j,l,k} (y_{17,i,j,l,k} - y_{18,i,j,l,k})] &= 0 \\
z_{n,i,j,l,k} - \frac{1}{A_{Tray}} [L2_{i,j,l,k} (x_{n-1_{i,j,l,k}} - x_{n,i,j,l,k}) - V_{i,j,l,k} (y_{n,i,j,l,k} - y_{n+1,i,j,l,k})] &= 0, \\
n &= 18, ..., 31 \\
z_{32,i,j,l,k} - \frac{1}{A_{Reboiler}} [L2_{i,j,l,k} x_{31,i,j,l,k} - (F - D_{l,k}) x_{32,i,j,l,k} - V_{i,j,l,k} y_{32,i,j,l,k}] &= 0 \\
x_{n,i,j,l,k} - x_{n,i-1,n_c,l,k} - h_{i,l,k} \sum_{m=1}^{n_c} \Omega_{m,j} z_{n,i,m,l,k} &= 0, \\
n &= 1, ..., 32 \\
x_{n,1,j,l,k} - x_n^0 - h_{i,l,k} \sum_{m=1}^{n_c} \Omega_{m,j} z_{n,1,m,l,k} &= 0, \\
n &= 1, ..., 32 \\
1 \leq u_{i,j,k} &\leq 5 \\
P_k X_k - P_k^d u_c &= 0
\end{aligned}\right\} \quad \begin{aligned} i &= 1, ..., n_f \\ j &= 1, ..., n_c \\ l &= 1, ..., N_k \\ k &= 1, ..., N \end{aligned}
$$

7.2 Parallel Timing Results for ESC and PCGSC Approaches

In order to show how the algorithms perform as the number of coupling variables increase, we generate 8 different problems, each containing 96 scenarios with

Table 7.1: Parameter and Variable Description

N	number of blocks
N_k	number of scenarios in each block
n_f	number of finite elements
n_c	number of collocation points in each finite element
X_k	vector of all the variables in each block
x_{Feed}	feed composition
u_c	vector of coupling variables across all the blocks
y_1^{set}	distillate measurements set points
u^{set}	control variables set points
α	weight for the distillate measurements
ρ	weight for the control variables
h	time step between two finite elements
y_n	vapor composition at the n^{th} stage.
x_n	liquid composition at the n^{th} stage.
z_n	derivative of x_n
u	vector of control variables
Ω	collocation coefficient matrix
$L1$	flow-rate of the liquid in the rectification section
V	vapor flow-rate in the column
$L2$	flow-rate of the liquid in the stripping section.
$\alpha_{A,B}$	relative volatility
F	feed flowrate
D	distillate flowrate
A_{Cond}	total molar holdup in the condenser
A_{Tray}	total molar holdup on each tray
$A_{Reboiler}$	total molar holdup in the reboiler

3 scenarios per block, for a total of 32 blocks. This format shows that the algorithm supports multiple scenarios on a single processor. Each scenario is created by selecting a random composition for the feed stream between 0.4 and 0.6.

To vary the number of coupling variables, we formulate the problem with a sequence of setpoint transitions. Fifty finite elements are assigned to a single transition, and the total number of transitions is varied from one to eight, producing problem formulations with coupling variables number 150 to 1200. Figure 7.2 shows the setting values and actual trajectories of y_1 for the distillation column problem with coupling variables number $150, 450, 750, 1050$. The input and output set-points for each transition are determined by simulation of the original model to guarantee that they are consistent with the desired steady-state behavior. In contrast with the QP problem from the previous section, in these studies, the overall problem size increases along with the number of coupling variables. Because the problem is different for each of the cases, the number of interior-point iterations required to solve the problems is different. In order to see the scalability of the algorithms independent of this effect, Table 7.2 shows the problem size and the average wall time per interior-point iteration for each of the case studies. For the parallel timing results, 32 processors were used.

For problems with more than 150 coupling variables, the memory requirements for the serial algorithms exceeded the hardware limitation of 12GB. Therefore, it was only possible to compute the speedup for the first entry in Table 7.2. As expected from the timing results for the previous example, the PCGSC-P algorithm scales much more favorably than the ESC-P algorithm as we increase the number of coupling variables.

Figure 7.3 shows the time spent in the specific components of the serial algorithms ESC-S and PCGSC-S for Case 1. In the serial Explicit-SC algorithm, the majority of

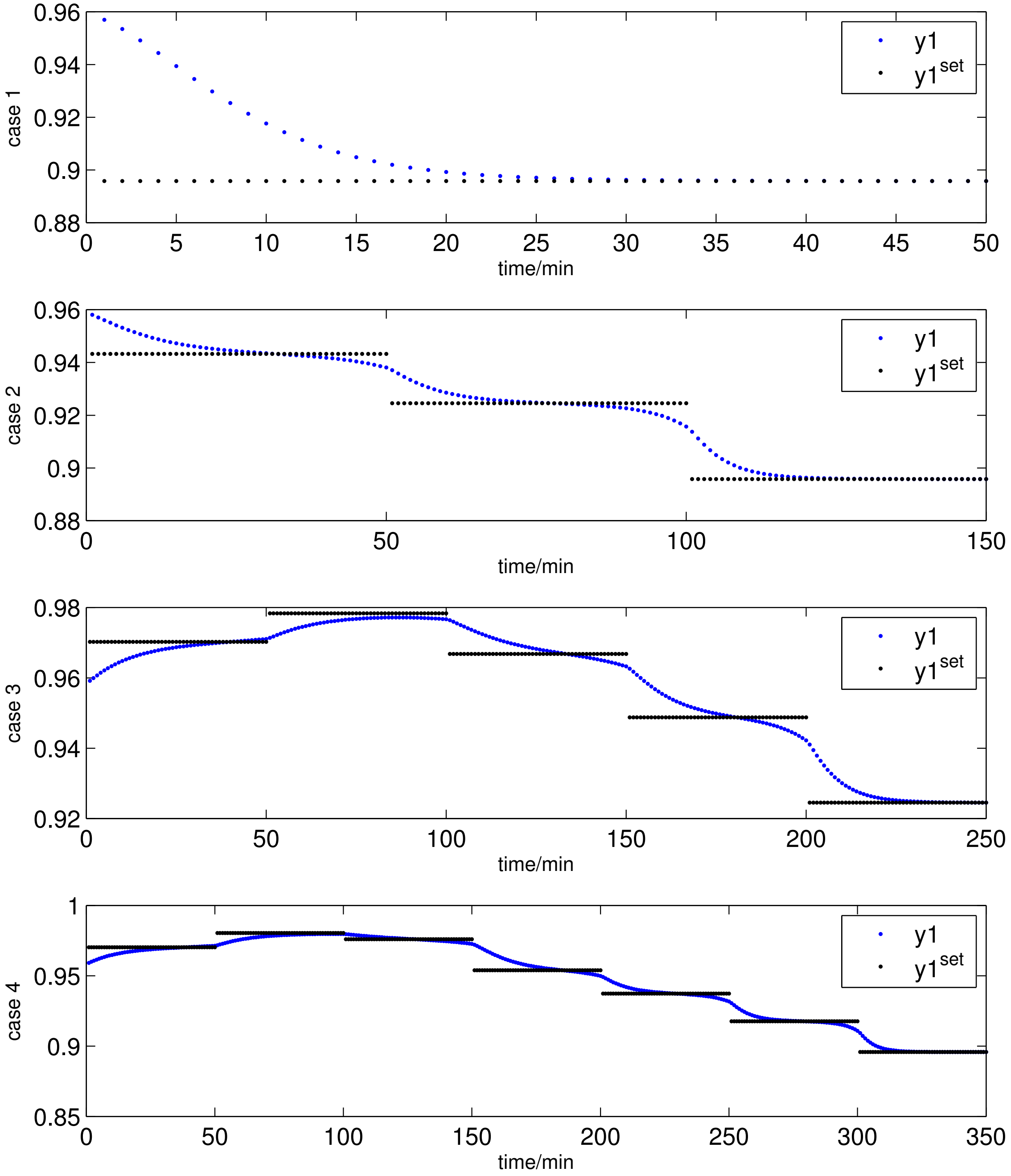


Figure 7.2: Setting values and actual trajectories of y_1 for the distillation column problem with coupling variables number $150, 450, 750, 1050$

Table 7.2: Wall Time per Iteration for Distillation Column Optimization

Case	# Vars.	# Coupling Vars.	FS-S time(s)	ESC-S time(s)	ESC-P time(s)	PCGSC-S time(s)	PCGSC-P time(s)
1	1430550	150	10.3	79.1	2.6	17.9	0.6
2	2861100	300	-	-	10.8	-	1.1
3	4291650	450	-	-	32.1	-	2.4
4	5722200	600	-	-	70.3	-	3.2
5	7152750	750	-	-	90.5	-	4.3
6	8583300	900	-	-	160.5	-	5.3
7	10013850	1050	-	-	218.0	-	6.3
8	11444400	1200	-	-	286.6	-	8.1

the time is spent in forming the Schur-complement. In the PCGSC-S algorithm, the computational time is significantly less because fewer backsolves are required to solve the Schur-complement. The wall clock time can be significantly reduced using parallel implementations since forming the Schur-complement and using PCG to solve the Schur-complement are highly parallelizable. The total time spent factorizing the block augmented matrix is slightly longer using the PCGSC algorithm than the Explicit-SC algorithm because 2 additional interior-point iterations were necessary using the PCGSC algorithm.

It is also important to investigate both the weak and strong scaling properties of our implementation of the algorithm for a fixed number of coupling variables. Here, we investigate four different cases, all with 1200 coupling variables. The number of scenarios in each case is 16, 32, 64, and 128 scenarios, respectively, and we assign one scenario to a block. Figure 7.4a shows weak-scaling results for the PCGCS-P approach. In these results, the abscissa shows the number of blocks and the number of processors used, while the ordinate shows the wall-clock time scaled by the time required for 16 blocks with 16 processors. Again, we see very little change

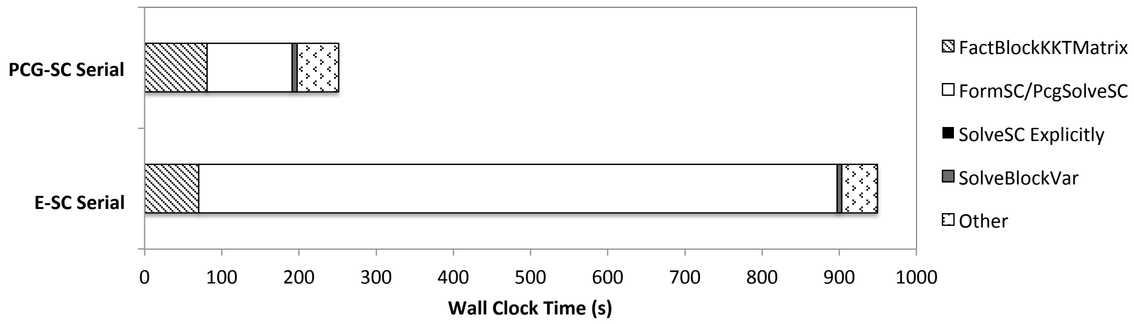
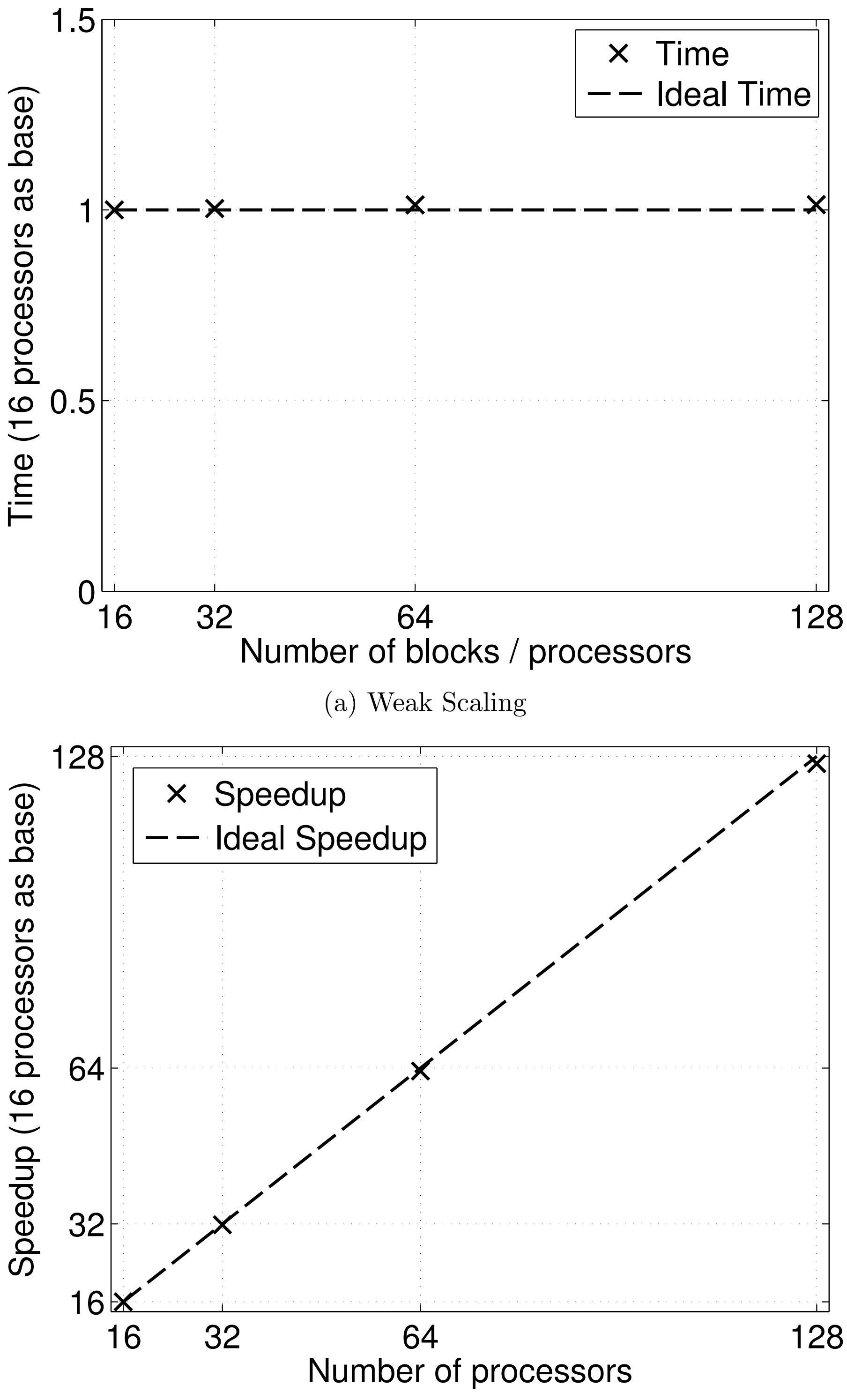


Figure 7.3: Wall clock time spent in specific components of the ESC-S and PCGSC-S for the distillation column with 150 coupling variables

in the solution time as we increase the number of blocks and processors up to 128. Figure 7.4b shows the strong scaling results for PCGSC-P. Here, the problem itself is fixed (128) scenarios, and the abscissa shows the number of processors used to solve the problem. The ordinate shows the speedup of the approach, scaled against the time required for 16 processors. Ideal, or perfect linear, scaling would produce speedups that lie on the dashed line. The PCGSC-P approach shows very little parallel performance deterioration on the column problem up to 128 processors.



(a) Weak Scaling

(b) Strong Scaling

Figure 7.4: Weak and strong scaling results for the PCGSC-P approach on the column example.

8. SUMMARY, CONCLUSIONS, AND FUTURE WORK*

Given the demand for efficient optimization of large nonlinear systems, coupled with the change in focus of hardware manufacturers towards multi-core and parallel architectures, there is a distinct need for development of effective parallel algorithms for solution of NLP problems. Fortunately, large-scale problems frequently possess inherent structure arising from distinct problem classes, and this structure can often be exploited in developing parallel solution strategies.

In this dissertation, we address parallel solution of block-structured NLP problems with complicating or coupling variables. We base our solution strategy on nonlinear interior-point methods, and develop parallel techniques for the linear algebra operations required by the algorithm. The dominant computational expense is the solution of the augmented system at each iteration of the interior-point algorithm. The explicit Schur-complement decomposition is a well-known strategy for exploiting the structure of these block angular problems and allowing for solution in parallel. However, as the number of coupling variables is increased, the computational time required to form and factorize the Schur-complement becomes high, eroding parallel performance.

While it is possible to solve the dense Schur-complement in parallel, using either

*Part of this section is reprinted with permission from "An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition" by Kang, J., Cao, Y., Word, D., and Laird, C.D., 2014. In: Computers and Chemical Engineering 71 (2014), pp 563-573, Copyright 2014 by Elsevier.

Part of this section is reprinted with permission from "Parallel solution of nonlinear contingency-constrained network problems" by Kang, J., Siirola, J.D., Watson J. and Laird, C.D., 2014. In: Proceedings of the 8th International Conference on Foundations of Computer-Aided Process Design FOCAPD 2014, July 13-17, 2014, Cle Elum, Washington, USA, Copyright 2014 Elsevier B.V.

Part of this section is reprinted with permission from "Nonlinear programming strategies on high-performance computers" by Kang, J., Chiang, N., Laird, C.D., and Zavala, V.M., 2015. In: Proceedings of the 54th IEEE Conference on Decision and Control, 2015, Osaka, Japan, Copyright 2015 Elsevier B.V.

direct or iterative approaches, for large systems, even the memory requirement for the Schur-complement alone can become prohibitive. We seek instead to avoid the need to form the Schur-complement at all. The implicit Schur-complement approach developed in this work uses a preconditioned conjugate gradient method to solve the Schur-complement system where only matrix-vector products across the Schur-complement are required. The range of available preconditioners is limited since the Schur-complement is never explicitly formed. Here, we use the automatic preconditioning technique of Morales and Nocedal (2000), which is based on a limited-memory BFGS (L-BFGS) update.

We first demonstrate the performance of explicit Schur-complement decomposition on the contingency-constrained ACOPF problems. Section 5 presents a rectangular IV formulation for the contingency-constrained ACOPF problem and demonstrates that the parallel Schur-complement based, nonlinear interior-point method described in Kang et al. (2014) can dramatically reduce solution times for this problem. Moreover, a comparison of explicit Schur-complement decomposition and Progressive Hedging approach on the same problem is executed to show the performance difference between external and internal partitioning methods. Our results show that the explicit Schur-complement decomposition approach is effective for problems with a small number ($\sim$100) of coupling variables.

We then demonstrate the performance of the PCG implicit Schur-complement decomposition approach on a set of parameter estimation problems and a set of nonlinear dynamic distillation column problems. The PCG implicit Schur-complement decomposition approach developed in this dissertation outperformed the explicit Schur-complement approach on every test case studied, providing significantly improved solution times over the explicit approach as the number of coupling variables is increased. Furthermore, the algorithm showed excellent weak and strong scaling properties in studies up to 128 processors.

These results show significant promise, and future work for this project is focused on producing a reliable implementation to release for other scientists and engineers. We plan to investigate the performance of different preconditioners (e.g., under varying condition number), and to explore the algorithm performance and tuning on additional problems. (E.g., how many update pairs should be stored in the L-BFGS update?)

For future work on the power grid problems, we will also address a much larger transmission network from the Matpower test suite (with approximately 3000 buses). With a larger number of generators, we suspect that the implicit PCG approach introduced in (Kang et al., 2014) will be much more effective. Finally, while the explicit Schur-complement approach yields significant speedup, the parallel efficiency is much lower than we have seen in other problems. This can be explained by the difference in factorization time for individual scenarios based on which transmission element was removed. We will also seek to address this load balancing issue and increase parallel efficiency.

A broad list of recommended research topics in this area is given below:

- These methods have seen little use outside the expert research community. Improvements are still necessary in modeling languages and implementation details such as ease of installation on high-performance computing software.

- Emerging architectures like the GPU provide potential for massively parallel computations at relatively low cost. However, the SIMD nature of these architectures makes it significantly more challenging to implement effective parallel algorithms. More research is necessary to determine effective ways of utilizing these architectures for parallel solution of general and block-structured NLP problems.

- Iterative linear solvers provide a natural framework for parallel algorithms, and they are highly appropriate for SIMD architectures. While there has been significant work on the use of iterative methods for the augmented system, for problems of general structure (or general structure within blocks), effective preconditioners are difficult to find. These methods have not been as successful as direct factorization methods based on block decomposition, and more research in this area is required.

- The majority of research in this area has focused on block-bordered diagonal structures like those arising with primal and dual coupling. More research is necessary for other common structures, including those arising from time-discretized systems and network-structured problems.

8.1 A Listing of the Major Contributions of this Dissertation Work

- **Implementation of an interior-point framework that provides an extensive foundation for new decomposition algorithms:** In this work, we have implemented a set of foundational classes for building decomposition algorithms for linear algebra. This framework has been extended to include several important classes for nonlinear programming. The several variants implemented based on this framework serve as excellent extension examples, and current and former graduate students have used this framework to develop new decomposition algorithms without the need to develop their own interior-point method (Word et al., 2014).

- **Formalizing the inertia requirement for Schur-complement decomposition algorithms in the context of nonlinear interior-point methods**: Although many scholars have worked on similar decomposition approaches, the majority of research in this area focus on linear and quadratic problems and do not address nonlinear problems. The inertia requirement is an essential part for solving nonlinear, nonconvex optimization problems, and this dissertation formalizes the inertia requirement in the context of the decomposition approaches.

- **Development of the necessary theory and implementation for *implicit* solution of the Schur-complement system with an iterative linear solver**: There has been extensive work on methods to try to reduce the cost of forming and factorizing (or solving) the Schur-complement. In this research, we demonstrate that this system can be solved implicitly, avoiding the need to form and factorize all-together. This approach brings significant performance benefits.

- **Development of a highly efficient (in terms of strong and weak scaling) decomposition approach for structured nonlinear programming problems with primal coupling**: In addition to the theoretical development of new algorithms, this dissertation work provides an efficient parallel implementation written in C++ that demonstrates highly efficient execution.

- **Efficient and reliable solution of nonlinear contingency constrained ACOPF problems**: This work addresses an important problem in the area of optimal power grid management. While existing techniques typically solve a linearized model and then test the solution for feasibility in the nonlinear case, this work shows that optimization of this important optimization under uncertainty problem is possible in a real-time context.

REFERENCES

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pages 483–485. ACM.

Amestoy, P. R., Duff, I. S., and L'Excellent, J.-Y. (2000). Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184(2):501–520.

Amestoy, P. R., Duff, I. S., LExcellent, J.-Y., and Koster, J. (2001). MUMPS: a general purpose distributed memory sparse solver. In *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, pages 121–130. Springer.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., and Zhang, H. (2014). PETSc Web page. http://www.mcs.anl.gov/petsc.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., and Zhang, H. (2015). PETSc Users Manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory.

Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F. (1997). Efficient management of parallelism in object oriented numerical software libraries. In Arge, E., Bruaset, A. M., and Langtangen, H. P., editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press.

Bergamaschi, L., Gondzio, J., and Zilli, G. (2004). Preconditioning indefinite systems in interior point methods for optimization. *Computational Optimization and*

Applications, 28:149–171.

Biegler, L. T. (2010). *Nonlinear programming: concepts, algorithms, and applications to chemical processes*, volume 10. Society for Industrial and Applied Mathematics.

Biros, G. and Ghattas, O. (2005). Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization. Part I: The Krylov-Schur solver. *SIAM J. Sci. Comput.*, 27:687–713.

Borges, C. and Alves, J. (2007). Power system real time operation based on security constrained optimal power flow and distributed processing. In *Power Tech, 2007 IEEE Lausanne*, pages 960 – 965.

Cain, M. B., O'Neill, R. P., and Castillo, A. (2012). History of optimal power flow and formulations. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-1-history-formulation-testing.pdf.

Calafiore, G. C. and Fagiano, L. (2013). Stochastic model predictive control of lpv systems via scenario optimization. *Automatica*, 49(6):1861–1866.

Campaigne, C., Lipka, P. A., Pirnia, M., O'Neill, R. P., and Oren, S. (2013). Testing stepsize limits for solving the linearized current voltage AC optimal power flow. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-9-stepsizelimits-iliv-acopf.pdf.

Cao, Y., Laird, C. D., and Zavala, V. M. (2015a). Clustering-based preconditioning for stochastic programs. *submitted to Computational Optimization and Applications.*

Cao, Y., Seth, A., and Laird, C. D. (2015b). A parallel augmented lagrangian interior-point approach for large-scale NLP problems on graphics processing units. *submitted to Computers and Chemical Engineering.*

Capitanescu, F., Glavic, M., Ernst, D., and Wehenkel, L. (2006). Applications of security-constrained optimal power flows. In *Proceedings of Modern Electric Power Systems Symposium, MEPS06.*

Castillo, A. and O'Neill, R. P. (2013a). Computational performance of solution techniques applied to the ACOPF. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-5-computational-testing.pdf.

Castillo, A. and O'Neill, R. P. (2013b). Survey of approaches to solving the ACOPF. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-4-solution-techniques-survey.pdf.

Castro, J. (2007). An interior-point approach for primal block-angular problems. *Computational Optimization and Applications*, 36(2-3):195–219.

Chiang, N., Petra, C. G., and Zavala, V. M. (2014). Structured nonconvex optimization of large-scale energy systems using PIPS-NLP. In *Proc. of the 18th Power Systems Computation Conference (PSCC), Wroclaw, Poland.*

Colombo, M., Grothey, A., Hogg, J., Woodsend, K., and Gondzio, J. (2009). A structure-conveying modelling language for mathematical and stochastic programming. *Mathematical Programming Computation*, 1(4):223–247.

Dollar, H. S. (2007). Constraint-style preconditioners for regularized saddle point problems. *SIAM Journal on Matrix Analysis & Applications*, 29(2):672–684.

Feng, Y., Rios, I., Ryan, S. M., Spürkel, K., Watson, J.-P., Wets, R. J.-B., and Woodruff, D. L. (2015). Toward scalable stochastic unit commitment. Part 1: load scenario generation. *Energy Systems*, pages 1–21.

Forsgren, A., Gill, P., and Wright, M. (2002). Interior methods for nonlinear optimization. *SIAM Review*, 44(4):525–597.

Forsgren, A., Gill, P. E., and Griffin, J. D. (2007). Iterative solution of augmented systems arising in interior methods. *SIAM Journal on Optimization*, 18:666–690.

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

Gade, D., Hackebeil, G., Ryan, S., Watson, J., Wets, R., and Woodruff, D. (2014). Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. *Under review*.

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

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

Grothey, A. and Qiang, F. (2013). PSMG: A parallel problem generator for structure conveying modelling language for mathematical programming. *presentation at ICCOPT*, 2009.

Gupta, A. (2000). WSMP: Watson sparse matrix package (Part-I: direct solution of symmetric sparse systems). *IBM TJ Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC*, 21886.

Hahn, J. and Edgar, T. F. (2002). An improved method for nonlinear model reduction using balancing of empirical gramians. *Computers and Chemical Engineering*, 26(10):1379–1397.

Hart, W., Laird, C., Watson, J., and Woodruff, D. (2012). *Pyomo: Optimization Modeling in Python*, volume 67. Springer-Verlag New York.

Hart, W. E., Watson, J.-P., and Woodruff, D. L. (2011). Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260.

Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., et al. (2005). An overview of the trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423.

Horton, R. R. (1987). *Multivariable control of an energy-integrated distillation column*. PhD thesis, Univ. of Texas at Austin.

HSL (2011). A collection of Fortran codes for large scale scientific computation. `http://www.hsl.rl.ac.uk`.

Huang, R. and Biegler, L. T. (2009). Robust nonlinear model predictive controller design based on multi-scenario formulation. In *Proc. of the American Control Conference*, pages 2341–2342.

Huchette, J., Lubin, M., and Petra, C. (2014). Parallel algebraic modeling for stochastic optimization. In *Proceedings of the First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 29–35. IEEE Press.

Kang, J., Cao, Y., Word, D. P., and Laird, C. (2014). An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition. *Computers & Chemical Engineering*, 71:563–573.

Kuzmin, A., Luisier, M., and Schenk, O. (2013). Fast methods for computing selected elements of the greens function in massively parallel nanoelectronic device simulations. In Wolf, F., Mohr, B., and Mey, D., editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 533–544. Springer Berlin Heidelberg.

Laird, C. D. and Biegler, L. T. (2008). Large-scale nonlinear programming for multi-scenario optimization. In *Modeling, Simulation and Optimization of Complex Processes*, pages 323–336. Springer.

Lipka, P. A., O'Neill, R. P., and Oren, S. (2013). Developing line current magnitude constraints for IEEE test problems. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-7-line-constraints.pdf.

Lubin, M. and Dunning, I. (2015). Computing in operations research using Julia. *INFORMS Journal on Computing*, 27(2):238–248.

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

Lubin, M., Petra, C. G., Anitescu, M., and Zavala, V. (2011). Scalable stochastic optimization of complex energy systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–10. IEEE.

Morales, J. L. and Nocedal, J. (2000). Automatic preconditioning by limited memory Quasi-Newton updating. *SIAM J. OPTIM.*, 10:1079–1096.

Morales, J. L. and Nocedal, J. (2001). Algorithm 809: PREQN: Fortran 77 subroutines for preconditioning the conjugate gradient method. *ACM Transactions on Mathematical Software*, 27(1):83–91.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. New York: Springer.

Oliveira, A. and Sorensen, D. (2005). A new class of preconditioners for large-scale linear systems from interior point methods for linear programming. *Linear Algebra and its Applications*, 394:1–24.

O'Neill, R. P., Castillo, A., and Cain, M. (2012a). The computational testing of AC optimal power flow using the current voltage formulations. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-3-iv-linearization-testing.pdf.

O'Neill, R. P., Castillo, A., and Cain, M. (2012b). The IV formulation and linear approximations of the AC optimal power flow problem. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-2-iv-linearization.pdf.

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

Petra, C. G., Schenk, O., Lubin, M., and Gertner, K. (2014). An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization. *SIAM Journal on Scientific Computing*, 36(2):C139–C162.

Phan, D. and Kalagnanam, J. (2012). Distributed methods for solving the security-constrained optimal power flow problem. In *Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES*, pages 1 – 7.

Pirnia, M., O'Neill, R. P., Lipka, P. A., and Campaigne, C. (2013). A computational study of LP approximation to the IV ACOPF formulation. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-8-preprocessed-constraints-iliv-acopf.pdf.

Poulson, J., Marker, B., Van de Geijn, R. A., Hammond, J. R., and Romero, N. A. (2013). Elemental: a new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software (TOMS)*, 39(2):13.

Qiu, W., Flueck, A., and Tu, F. (2005). A new parallel algorithm for security constrained optimal power flow with a nonlinear interior point method. In *Power Engineering Society General Meeting, 2005. IEEE*, volume 1, pages 447 – 453.

Rockafellar, R. T. and Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA.

Schecter, A. and O'Neill, R. P. (2013). Exploration of the ACOPF feasible region for the standard IEEE test set. FERC Staff Technical Paper. http://www.ferc.gov/industries/electric/indus-act/market-planning/opf-papers/acopf-6-test-problem-properties.pdf.

Schenk, O., Bollhöfer, M., and Römer, R. A. (2008a). On large-scale diagonalization techniques for the anderson model of localization. *SIAM Rev.*, 50(1):91–112.

Schenk, O., Wächter, A., and Hagemann, M. (2007). Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. *Computational Optimization and Applications*, 36(2-3):321–341.

Schenk, O., Wächter, A., and Weiser, M. (2008b). Inertia-revealing preconditioning for large-scale nonconvex constrained optimization. *SIAM Journal on Scientific Computing*, 31(2):939–960.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. http://www.cs.cmu.edu/ quakepapers/painless-conjugate-gradient.ps.

Sylvester, J. J. (1952). A demonstration of the theorem that every homogeneous quadratic polynomial is reducible by real orthogonal substitutions to the form of a sum of positive and negative squares. *Philosophical Magazine IV*, pages 138–142.

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

Word, D. P., Cummings, D. A., Burke, D. S., Iamsirithaworn, S., and Laird, C. D. (2012). A nonlinear programming approach for estimation of transmission parameters in childhood infectious disease using a continuous time model. *Journal of*

The Royal Society Interface, 9(73):1983–1997.

Word, D. P., Kang, J., Akesson, J., and Laird, C. D. (2014). Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Computational Optimization and Applications*, 59(3):667–688.

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

Zhu, Y. and Laird, C. (2008). A parallel algorithm for structured nonlinear programming. In *Proceeding of 5th International Conference on Foundations of Computer-Aided Process Operation, FOCAPO*, pages 345–348.

Zhu, Y., Legg, S., and Laird, C. (2011). Optimal operation of cryogenic air separation systems with demand uncertainty and contractual obligations. *Chemical Engineering Science*, 66(5):953–963.

Zimmerman, R. D. and Murillo-S'anchez, C. E. (2011). *Matpower 4.1 User's Manual.* Power Systems Engineering Research Center (Pserc).