2022

Efficient Numerical Optimization for Parallel Dynamic Optimal Power Flow Simulation Using Network Geometry

Rylee Sundermann
South Dakota State University, rylee.sundermann@gmail.com

EFFICIENT NUMERICAL OPTIMIZATION FOR PARALLEL DYNAMIC OPTIMAL POWER FLOW SIMULATION USING NETWORK GEOMETRY

BY

Rylee Sundermann

A thesis submitted in partial fulfilment of the requirements for the

Masters of Science

Major in Mathematics

South Dakota State University

2022

THESIS ACCEPTANCE PAGE

Rylee Sundermann

This thesis is approved as a creditable and independent investigation by a candidate for the master's degree and is acceptable for meeting the thesis requirements for this degree. Acceptance of this does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department.

Jung-Han Kimn
Advisor Date

Kurt Cogswell
Department Head Date

Nicole Lounsbery, PhD
Director, Graduate School Date

I dedicate this work to my wife Tessa Sundermann. Without whom I would not be where I am today.

ACKNOWLEDGEMENTS

I would like to thank my committee members Dr. Donald Vestal, Dr. Zhiguang Wang, and my advisor Dr. Jung-Han Kimn.

I would also like to thank Dr. Hong Zhang from Argonne National Laboratory and Dr. Shrirang Abhyankar from Pacific Northwest National Laboratory for the guidance and help they provided. This project would not have been possible without their assistance.

Thank you to Dr. Tim Hansen for introducing me to the power flow problem, and meeting with me throughout my research.

I would like to thank Dr. Jung-Han Kimn for taking a chance on me as a freshman and introducing me to research. Working on projects with him has been truly remarkable.

Thank you to my wife Tessa Sundermann for all the love and support throughout the years.

A final thank you to all who read this.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

EFFICIENT NUMERICAL OPTIMIZATION FOR PARALLEL DYNAMIC OPTIMAL POWER FLOW SIMULATION USING NETWORK GEOMETRY

Rylee Sundermann

2022

In this work, we present a parallel method for accelerating the multi-period dynamic optimal power flow (DOPF). Our approach involves a distributed-memory parallelization of DOPF time-steps, use of a newly developed parallel primal-dual interior point method, and an iterative Krylov subspace linear solver with a block-Jacobi preconditioning scheme. The parallel primal-dual interior point method has been implemented and distributed in the open-source PETSc library and is currently available. We present the formulation of the DOPF problem, the developed primal dual interior point method solver, the parallel implementation, and results on various multi-core machines. We demonstrate the effectiveness our proposed block-Jacobi preconditioner and various Krylov subspace methods at improving parallel performance.

1 INTRODUCTION

While maintaining a reliable power grid there is an innate goal to reduce cost. Towards this goal several models to optimize the power distribution across the power grid have been developed.

The optimization on the power grid with a set load is called AC-Optimal Power Flow or (ACOPF). This method can be expanded to consider multiple moments of different loads, resulting in the dynamic optimal power flow (DOPF), alternatively referred to as the multi-period ACOPF (MPOPF) problem. This allows the DOPF to find the optimal operation across a set time horizon while subject to the network constraints. The DOPF problem over a discretized time horizon is essentially a series of ACOPF problems with an inter-temporal connection (Figure 1). The complexity of the DOPF

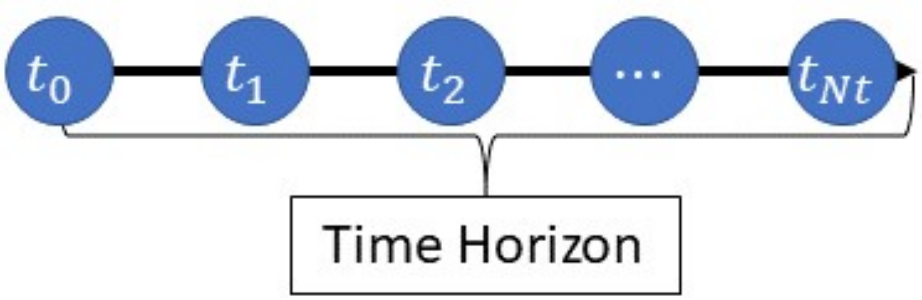

Figure 1: Visualization of DOPF as a series of ACOPF problems along a time horizon.

problem lies with the computation and memory demands as the time horizon expands. Memory limitations are a significant limiting factor for a single processor on most time horizons of the Texas 2000-bus power grid, a visualization of which is presented in Fig. 2. To address these limiting factors, we built both a parallel DOPF solver and a generalized parallel primal dual interior point method optimization solver. The parallel DOPF solver is built upon the the Exascale Grid Optimization toolkit (ExaGO) [2] from Pacific Northwest National Laboratory due to its hierarchical structure and foundation utilizing the Portable, Extensible Toolkit for Scientific Computation (PETSc) [9] data management

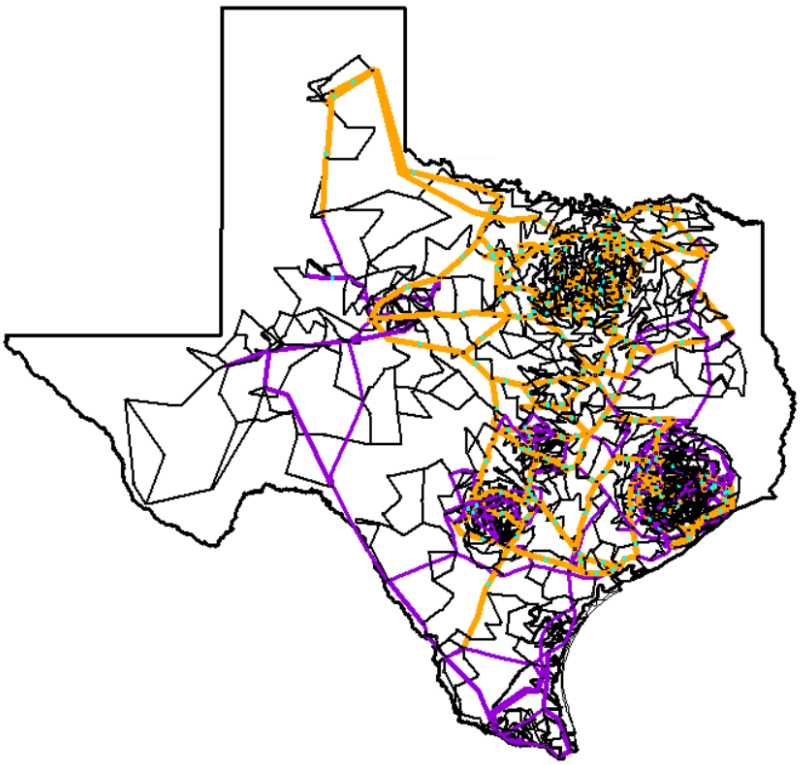

Figure 2: ACTIVSg2000 network for a single time-step.
Source: [10]

object DMNetwork [1] which will be discussed in section 4. The structure of ExaGO allows the DOPF solver to build upon the the already implemented ACOPF solver, and the utilization of the PETSc library allows parallel vector and matrix creation and hosts a suite of parallel solvers designed to facilitate parallel implementation. These PETSc objects allow for a direct link to PETSc's sublibrary Toolkit for Advanced Optimization (TAO) which hosts our parallel primal dual interior point method (PDIPM).

Within numerical optimization, primal dual interior point methods are a subset of interior point methods that find optimal solutions to linear/nonlinear convex optimization problems by traversing the feasible region. The method minimizes the function $f(x)$ that is subject to equality $G(x)$ and inequality $H(x)$ constraints. To create a Lagrangian function we rewrite the inequality constraints as equality constraints by introducing a barrier function. Thus, we minimize $f(x) - \mu \sum_{i=1}^{NH} ln(z_i)$ subject to the equality

constraints $G(x)$ and $H(x) - z$ where $NH = dim(H(x))$. Thus the Lagrangian function

$$L_\mu(X) = f(x) + \lambda_G^T G(x) - \lambda_H^T (H(x) - z) - \mu \sum_{i=1}^{NH} ln(z_i)$$

can be minimized using a Newtons method solver.

1.1 LITERATURE REVIEW

With the DOPF problem being comprised of a series of ACOPF problems the complexity in terms of memory and computation grows almost linearly with time steps. Due to this, the greatest challenge for the solution of over long time horizons is memory and computational efficiency. There are several different methods to combat these challenges and they can all be separated into sequential and parallel methods.

1.1.1 SEQUENTIAL DOPF

Within DOPF solvers, the sequential solvers primarily run into memory bottlenecks and thus tend to solve smaller network problems. An example of this is BATTPOWER [20], a DOPF solver built upon the commonly used ACOPF and power flow (PF) solver MATPOWER. This work presented a memory efficient approach to solving the DOPF problem by refactoring the system to take advantage of the sparse nature of the problem. While this does improve memory usage, the process of applying a block-Jacobi preconditioner to the parallel PDIPM accomplishes the same result while obtaining super-linear scalability on the Hessian and gradient evaluations. These evaluations are the most computationally expensive aspect of their simulation.

In [21], it was demonstrated on the 3-bus cases that utilizing analytical derivatives one can achieve a reduced computation time as compared to numerical differentiation. This result was achieved by comparing a commercial solver that utilized numerical differentiation to a designed PDIPM solver However, as with BATTPOWER, these

analytical derivatives are the most computationally expensive aspect of their simulation. This is a contrast from our research, in which these analytical Hessian and gradient evaluations compose only 5% of total computation time.

Within ExaGO [3], the library this work is built upon, IPOPT [18] is utilized to solve the DOPF problem. In Section 5, we will directly compare our parallel results to the results obtained by this work and show the benefits of our work in building a parallel PDIPM and DOPF simulation.

1.1.2 PARALLEL DOPF

There are a few groups working on a parallel DOPF solver, including a parallel multi-period contingency constrained solver. The work presented in [15] consisted of a parallel solver built in Julia and utilized the parallel interior point method solver PIPS [13]. The limitation of this approach comes when trying to solve large network systems. The LDL^T factorization PIPS uses is expensive due to communication and memory requirements. While our work utilizes the same factorization, we take advantage of the structure of the DOPF problem and implement a block-Jacobi preconditioner. This allows for the application of the LDL^T factorization to be applied on each processor independently reducing both memory and communication. Additionally, our work expands on this process with the introduction of a general parallel primal dual interior point method solver as PIPS requires the optimization problem to be of a specific structure.

In the results of [12], they describe a parallel approach to the DOPF problem that is constructed on a single processor then distributed using Middleware Software to a parallel machine. Their proposed Genetic Algorithm based DOPF appears to be unscalable for large scale HPC simulations as it failed to converge in their tests in their case of 200 generators.

Within the tech report [16], this group utilized PETSc for their parallel solve of the

KKT problem. However, the formulation of the problem starts initially in MATLAB using MATPOWER then the solve calls their C++ code which then distributes the matrix across processors before solving the system. The limitations of this process are the memory of a single processor and the communication bandwidth between processors. Additionally this only parallelizes the solve of the KKT system and not the construction leading to a sequential construction of each time step. These limitations of only parallelizing the solution step also occur in [12] previously described. Finally the proposed Additive Schwarz method (ASM) preconditions the KKT matrix using the data of two time steps that is not stored on that processor, adding to the required communication. Additionally as the underlying network expands the memory required for this will increase. For this reason we believe that our block-Jacobi approach which does not require overlap yet takes advantage of the sparse inter-temporal connections is a more desirable preconditioning method for this application.

In [14] the same algorithm is used as in [15]. Within this paper, the largest network solved was the 1354-bus, however they presented solution times of the linear system for the 9241-bus system. Thought the 9241-bus system had both memory and convergence issues. This highlights the benefits of our block-Jacobi preconditioner, as they apply more processors per time-step allowing for a greater amount of total memory as compared to our approach, however, due to the memory and communication bandwidth the LDL^T factorization it is inefficient. We demonstrate this limitation of the LDL^T factorization in Section 5.

1.2 ORGANIZATION OF THE WORK

This work begins with the mathematical formulation of the Dynamic Optimal Power Flow (DOPF) problem followed by a description of numerical optimization focused on Primal Dual Interior Point Methods (PDIPM) in sections 2 and 3 respectively. Then in section 4 an introduction to parallel programming, packages used and the parallel implementation.

Culminating in a description of numerical results on multiple parallel computers in section 5.

2 DYNAMIC OPTIMAL POWER FLOW

The dynamic optimal power flow (DOPF) problem aims to optimize the power production by minimizing the cost of production over a time horizon. Toward this end, we discretize the time horizon creating what are essentially, snapshots of the power grid. Directly solving the alternating current optimal power flow (ACOPF) problem for each of the snapshots independently, however, can lead to physically impossible shifts in the network distribution. To counter this we impose linking conditions between the snapshots to maintain physically meaningful results. Adding this link requires the network to remain within physically feasible bounds but requires all systems to be solved simultaneously, significantly increasing the size of the optimization problem. In the subsequent sections we will discuss the mathematical formulation of the ACOPF problem, the inter-temporal linking constraints, and the DOPF problem.

2.1 FOMULATION OF ACOPF

Alternating Current Optimal Power Flow (ACOPF) aims to find the most cost effective production and distribution across the power grid at a set instance. The mathematical foundation of ACOPF is a minimization problem where the cost of power production varies across generators. For any generator G_k, there are scalars α, β, γ that form a quadratic cost function dependent on the power produced. Thus, the total cost function,

$$f(x) = \sum_{k=1}^{Ng} \left(\alpha_k P_{G_k}^2 + \beta_k P_{G_k} + \gamma_k\right) \tag{1}$$

is a sum of these costs over the number of generators Ng and the x vector is comprised of

$$x = \begin{bmatrix} Pg & Qg & V_R & V_I \end{bmatrix}^T \tag{2}$$

for each bus. V_R and V_I exist on every bus while Pg and Qg correlate to the generators on any given bus with the following constraints:

$$\begin{bmatrix} Pg^- & Qg^- & V_R^- & V_I^- \end{bmatrix}^T \leq \begin{bmatrix} Pg & Qg & V_R & V_I \end{bmatrix}^T \leq \begin{bmatrix} Pg^+ & Qg^+ & V_R+ & V_I+ \end{bmatrix}^T \quad (3)$$

where $Pg^{+/-}$ and $Qg^{+/-}$ are generator dependent while $V_R^{+/-}$ and $V_i^{+/-}$ are unbounded and are treated as $+\infty$ and $-\infty$ respectively. Furthermore, the feasibility region of this problem is restricted by physical constraints on the power network. The equality constraints:

$$\begin{aligned} \Delta P_f = & \sum_{A_{br}(f,t)=1} (G_{ff}(V_{Rf}^2 + V_{If}^2) + V_{Rf}(G_{ft}V_{Rt} - B_{ft}V_{It}) + V_{If}(B_{ft}V_{Rt} + G_{ft}V_{It})) \\ & - \sum_{A_G(f,k)=1} P_{Gk} + \sum_{A_L(f,j)\neq 0} (P_{Dj}) \\ & = 0, \end{aligned} \quad (4)$$

$$\begin{aligned} \Delta Q_f = & \sum_{A_{br}(f,t)=1} (-B_{ff}(V_{Rf}^2 + V_{If}^2) + V_{If}(G_{ft}V_{Rt} - B_{ft}V_{It}) - V_{Rf}(B_{ft}V_{Rt} + G_{ft}V_{It})) \\ & - \sum_{A_G(f,k)\neq 0} Q_{Gk} + \sum_{A_L(f,j)=1} (Q_{Dj}) \\ & = 0, \end{aligned} \quad (5)$$

$$\Delta\theta_{ref} = V_{Iref} - V_{Rref}\tan(\theta_{ref}) = 0. \quad (6)$$

where (4) and (5) are the real and reactive power balance equations and (6) holds the reference angle constant. The inequality constraints consisting of voltage magnitude:

$$(V_{min})^2 \leq V_{Ri}^2 + V_{Ii}^2 \leq (V_{max})^2, \quad (7)$$

and power flow between each bus:

$$0 \leq \left(P_{ft}^2 + Q_{ft}^2\right) \leq (S_{ft}^+)^2,$$
$$0 \leq \left(P_{tf}^2 + Q_{tf}^2\right) \leq (S_{ft}^+)^2. \tag{8}$$

where V_{min} and V_{max} are given physical limits, and S_{ft}/S_{tf} are a variable max for each line labeled RATE_A, RATE_B, RATE_C for normal, short-term, and emergency operations.

2.2 FORMULATION OF DOPF

With expanding ACOPF to the DOPF problem we expand the system over time. Solving the DOPF of a time frame T requires assessing the ACOPF problem Nt times at Δt intervals. Therefore, the DOPF problem is also a minimization problem with the objective function

$$f(x) = \sum_{t=1}^{Nt} \sum_{k=1}^{Ng} \left(\alpha_k P_{G_k}^2 + \beta_k P_{G_k} + \gamma_k\right) \tag{9}$$

and the new x vector

$$x = \begin{bmatrix} x_1 & x_2 & \dots & x_t & \dots & x_{Nt} \end{bmatrix}^T \tag{10}$$

is comprised of the sub-vectors x_t from each ACOPF problem.

Then, the constraints for DOPF consist of the same equality and inequality constraints for each sub-vector as the ACOPF. Thus, at time step t the x_t sub-vector is subject to the constraints listed in (3) – (8). The additional time dependent constraint is an inequality constraint that handles generator ramping. These ramping constraint equations are

$$P_G(t - \Delta t) - P_G(t) \leq r_{G_k} \Delta t \tag{11}$$

and

$$P_G(t) - P_G(t - \Delta t) \leq r_{G_k} \Delta t. \tag{12}$$

These constraints require consecutive time-steps to be within a threshold of generator output allowing for the generators to increase production to meet the new demand. Due to the ramping constraints there is a link in time between current and neighboring time-steps. The impact of the temporal link in the parallel solution will be discussed in a later section.

3 PRIMAL DUAL INTERIOR POINT METHOD

Within numerical optimization primal dual interior point methods is a subset of the interior point methods that are widely implemented for finding the optimal solution to linear and nonlinear convex optimization problems. The significant difference between interior point method and the more common simplex method for linear optimization lies in how they traverse the feasible region. The simplex method finds optimal solutions by traversing the boundaries of the feasible region while interior point methods traverse the interior of the constraint bounds.

3.1 FORMULATION OF PDIPM

In this section, we construct a PDIPM for solving constrained nonlinear optimization problems and implement the PDIPM in the PETSc library as an open-source parallel solver. Consider the described optimization problem for DOPF, with the objective function $f(x)$, equality constraints $g(x)$, inequality constraints $h(x)$, and upper and lower bounds x^- and x^+ on x. Then the optimization problem can be written in a compact form as

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0, \\ & h(x) \geq 0, \\ & x^- \leq x \leq x^+. \end{aligned} \tag{13}$$

From (13), we combine the constraints and the bounds to obtain

$$G(x) = \begin{bmatrix} g(x) \\ x - x_{eq} \end{bmatrix} \text{ and } H(x) = \begin{bmatrix} h(x) \\ x^+ - x \\ x - x^- \end{bmatrix},$$

where x_{eq} represents the set of x variables where $x^- = x^+$. We introduce a set of slack variables z and a logarithmic barrier function to the original objective function to ensure

the positivity of z. Then (13) is formulated as the new optimization problem:

$$\begin{aligned} \min_{x} \quad & f(x) - \mu \sum_{i=1}^{N_H} \ln(z_i) \\ \text{s.t.} \quad & G(x) = 0, \\ & H(x) - z = 0, \end{aligned} \tag{14}$$

where N_H denotes the number of inequality constraints in H. Note that μ is driven to zero during the optimization.

Define $X = \begin{bmatrix} x, \lambda_G, \lambda_H, z \end{bmatrix}^T$. Then the final transformation of (14) is the single Lagrangian function:

$$L_\mu(X) = f(x) + \lambda_G^T G(x) - \lambda_H^T (H(x) - z) - \mu \sum_{i=1}^{N_H} \ln z_i, \tag{15}$$

where λ_G and λ_H are Lagrangian multipliers for the equality and inequality constraints, respectively. Additionally, the solution to (14) will result in a saddle point in (15). Applying Newton's method with (15) as the target function, we aim to find a critical point x^* that will minimize our original function (13).

The minimizer X^* of (15) must satisfy KKT conditions [11], which allow an extension of Lagrangian multipliers to inequality constraints. We apply a Newton's method to refine an initial guess X_0 by

$$X_{n+1} = X_n + \alpha \Delta X, \tag{16}$$

in which the search direction ΔX is calculated by solving

$$K \Delta X = -F. \tag{17}$$

The symmetric KKT matrix $K = \nabla^2 L_\mu$ is

$$\begin{bmatrix} W_{xx} & \nabla G(x)^T & -\nabla H(x)^T & 0 \\ \nabla G(x) & 0 & 0 & 0 \\ -\nabla H(x) & 0 & 0 & I \\ 0 & 0 & I & \Lambda_H * Z^{-1} \end{bmatrix}. \tag{18}$$

The right-hand side vector is

$$F = \begin{bmatrix} W_x \\ G(x) \\ z - H(x) \\ \Lambda_H e - \mu * Z^{-1} e \end{bmatrix}, \tag{19}$$

where

$$W_x = \nabla f(x)^T + \nabla G(x)^T \lambda_G - \nabla H(x)^T \lambda_H, \tag{20}$$

$$W_{xx} = \nabla^2 f(x) + \nabla^2 G(x)^T \lambda_G - \nabla^2 H(x)^T \lambda_H, \tag{21}$$

e is a vector of ones, I is the identity matrix, and Z and Λ_H are square matrices with z and λ_H along their diagonals, respectively.

At the end of each iteration, we test the convergence of the Newton's solver by testing the norm of F which is separated between prime and dual components described as the residual norm

$$r = \sqrt{W_x^T W_x + (Z\Lambda_H e - \mu e)^T (Z\Lambda_H e - \mu e)} \tag{22}$$

and constraint norm (c-norm)

$$\tau = \sqrt{G(x)^T G(x) + (z - H(x))^T (z - H(x))}. \tag{23}$$

These norms are then used as convergence criteria. In our implementation, convergence is reached when the absolute c-norm and either absolute or relative residual tolerances are met.

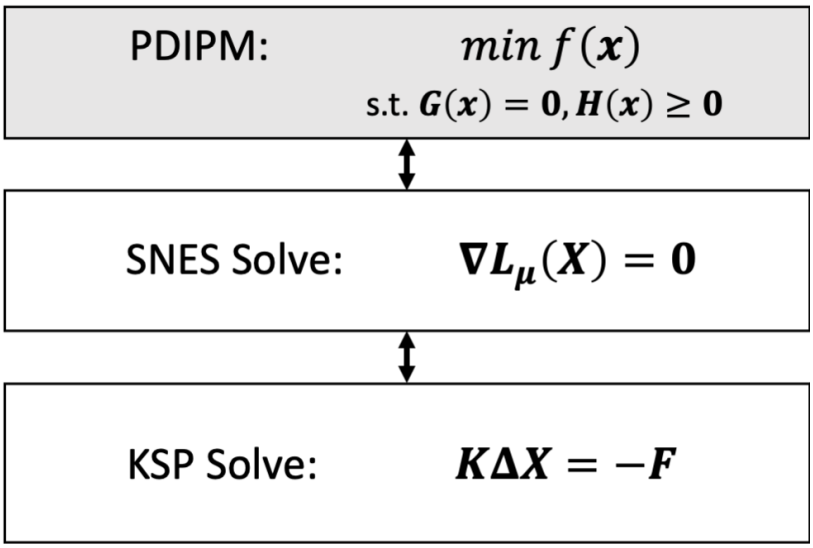

Figure 3: Software structure of PDIPM in PETSc
KSP: Krylov Subspace and Preconditioner methods.
SNES: Scalable Nonlinear Equation Solvers.

The PDIPM is implemented as a general constrained optimization solver in PETSc. Figure 3 illustrates the software structure of the PDIPM in PETSc. In this solver, we iteratively solve the KKT systems (17) using a preconditioned Krylov subspace method. PDIPM updates only the elements of $G(x)$, $H(x)$, $\nabla G(x)$, $\nabla H(x)$, and W_{xx} that directly depend on X_n.

The parallel implementation of the PDIPM method is achieved by distributing all involved vectors and matrices across multiple processors and utilizing PETSc scalable linear equation solvers (KSP) and Scalable Nonlinear Equation Solvers (SNES) [8].

4 PARALLEL PROGRAMMING

In this section we will discuss the various aspects of the parallel implementation of our code, including libraries used and the parallel distribution of the application. We will begin with a description of the PETSc library used for its vast array of parallel data structures and solvers, then describe its sub-library TAO where our PDIPM solver is implemented. Finally, we will conclude with a discussion of the parallelization of the ExaGO library from Pacific Northwest National Laboratory that is the foundation of our parallel DOPF solver.

4.1 PETSC LIBRARY

The Portable Extensible Toolkit for Scientific Computation (PETSc) library is an open-source repository comprised of data structures and routines built to facilitate parallel programming. The PETSc library is used internationally as the foundation of many parallel software packages, including ExaGO that our DOPF simulation is built on.

4.1.1 DATA STRUCTURES

ExaGO simulations utilize PETSc's DMNetwork data management object to contain the power grid connections and node data. For each time-step we create a DMNetwork object using the data from each instance to construct a snapshot of the grid at that moment, and use this to define the size of the full system. Additionally, they are used in parallel as each ACOPF calculation used DMNetwork's connection data to calculate (4), (5), and (8). This is accomplished by creating local work vectors for each network see Figure 4 for visualization. Finally, DMNetwork facilitates the local Vec to Array object transformations utilized within ExaGO's ACOPF to reduce calculation time (Fig. 4).

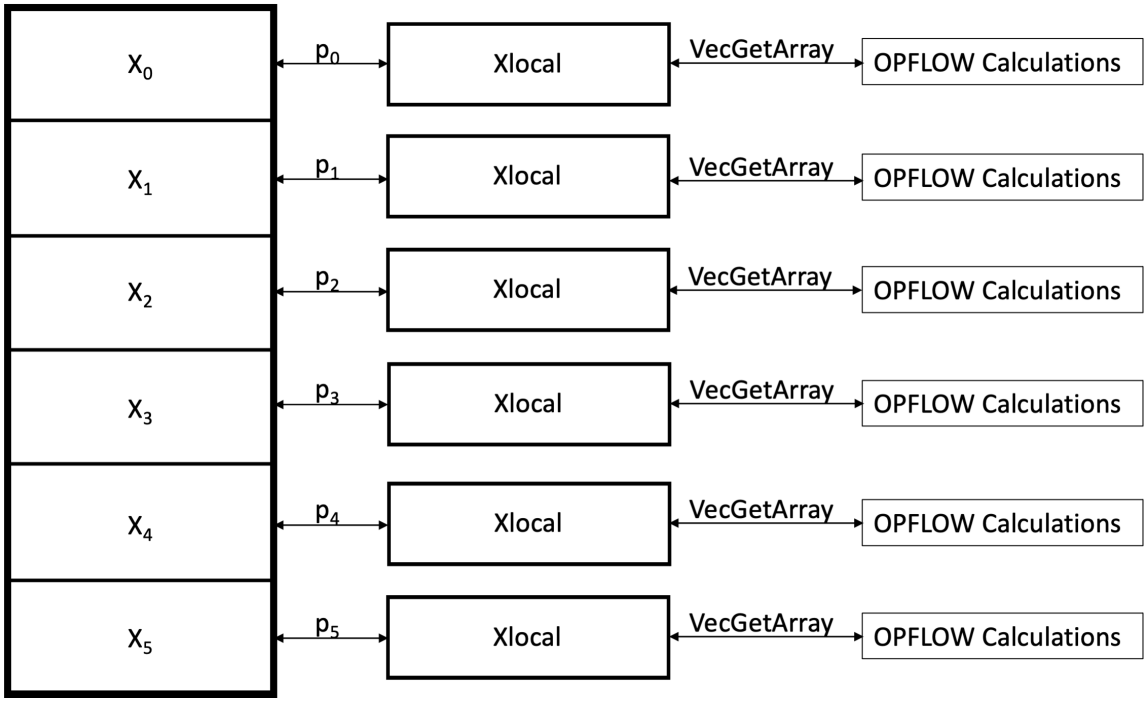

Figure 4: Global X vector to local x vector for 6 processors

4.1.2 SOLVERS

In our PDIPM solver within PETSc, we utilized two forms of parallel solvers: Scalable Nonliner Equation Solvers (SNES) and Krylov Subspace and Preconditioner (KSP) methods. We utilize the SNES solver to solve the nonlinear $\nabla L_\mu(X) = 0$ problem. We accomplish this by using the default Newton method solver with a line search. This non-linear Newton method has a linear sub-problem of the form

$$\nabla^2 L_\mu(X) * \Delta X = -\nabla L_\mu(X). \tag{24}$$

This linear sub-problem is solved utilizing a Krylov subpace method and a preconditioner.

For scaling data on both the 200bus and the 2000bus power girds (section 5), we utilized the default KSP Generalized minimal residual method with a sequential LDL^T, a parallel LDL^T and a block Jacobi preconditioner with LDL^T applied to inner diagonal sub-blocks. Additionally, we demonstrated performance of four additional Krylov

subspace methods that can be applied to our symmetric indefinite matrix. These four methods are a biconjugate gradient stabilized method (BCGS), a flexible biconjugate gradient stabilized method (FBCGS), a pipelined biconjugate gradient method (PIPEBCGS), and a generalized conjugate residual method (GCR).

4.2 TAO PDIPM

The toolkit of advanced optimization (TAO) is a software library built for large scale optimization problems. Built as a companion library to PETSc, TAO offers access to the full suite of parallel data structures and scalable linear solvers.In this section, we will describe the methodology of how the PDIPM method, which was described in section 3, is parallelized, improved with an inertia shift, and linked to our DOPF application.

4.2.1 PARALLEL IMPLEMENTATION

For the parallelization of PDIPM, there is an important distinction between global variables, which are labeled $Nx, Nh, etc.$, and the local variables, which are labeled $nx, nh, etc.$ This classification allows the algorithm to independently update and solve its part of the system. For example, when updating after an iteration each processor owns a portion of the global X vector, and it is more efficient to have it update the corresponding parts of KKT system than to communicate its information to another processor.

From the PDIPM solvers perspective, the user declares the sizes of all objects. For example, the variables Nx and Nh are based on the vectors passed in when the functions TaoSetInitialVector and TaoSetInequalityConstraintsRoutine are called. With these vectors, the local sizes for the system are defined based their distribution across processors. Thus, the partitions for vectors should match the row partitions of their matrices.

The local variables then define the distribution of the KKT system and its updating. For the following standard sequential KKT matrix,

$$K = \begin{array}{c} \\ \mathbf{Nx} \\ \mathbf{Ng} \\ \mathbf{Nh} \\ \mathbf{Nh} \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{Nx} & \mathbf{Ng} & \mathbf{Nh} & \mathbf{Nh} \end{array} \\ \begin{bmatrix} W_{xx} & \nabla G(x)^T & -\nabla H(x)^T & 0 \\ \nabla G(x) & 0 & 0 & 0 \\ -\nabla H(x) & 0 & 0 & I \\ 0 & 0 & I & \Lambda_H * Z^{-1} \end{bmatrix} \end{array}, \tag{25}$$

the data exists with all objects on a single processor with global variable-defined offsets. However, with multiple processors it is more complicated. Consider x to be split onto two processors with nx elements on each. Also, let nh and ng represent the number of element on each processor for the split $H(x)$ and $G(x)$. Then for Wxx, $\nabla G(x)$, $\nabla H(x)$ let the subscripts $11, 12, 21, 22$ represent which set the row/column is in. Finally, we will let $W = Wxx$. Then, the KKT matrix on two processors is

$$\begin{array}{c} \\ \mathbf{nx} \\ \mathbf{ng} \\ \mathbf{nh} \\ \mathbf{nh} \\ \mathbf{nx} \\ \mathbf{ng} \\ \mathbf{nh} \\ \mathbf{nh} \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccccc} \mathbf{nx} & \mathbf{ng} & \mathbf{nh} & \mathbf{nh} & \mathbf{nx} & \mathbf{ng} & \mathbf{nh} & \mathbf{nh} \end{array} \\ \begin{bmatrix} W_{11} & \nabla G(x)_{11}^T & -\nabla H(x)_{11}^T & 0 & W_{12} & \nabla G(x)_{12}^T & -\nabla H(x)_{12}^T & 0 \\ \nabla G(x)_{11} & 0 & 0 & 0 & \nabla G(x)_{12} & 0 & 0 & 0 \\ -\nabla H(x)_{11} & 0 & 0 & I & -\nabla H(x)_{12} & 0 & 0 & 0 \\ 0 & 0 & I & \Lambda_{H1} * Z_1^{-1} & 0 & 0 & 0 & 0 \\ W_{21} & \nabla G(x)_{21}^T & -\nabla H(x)_{21}^T & 0 & W_{22} & \nabla G(x)_{22}^T & -\nabla H(x)_{22}^T & 0 \\ \nabla G(x)_{21} & 0 & 0 & 0 & \nabla G(x)_{22} & 0 & 0 & 0 \\ -\nabla H(x)_{21} & 0 & 0 & 0 & -\nabla H(x)_{22} & 0 & 0 & I \\ 0 & 0 & 0 & 0 & 0 & 0 & I & \Lambda_{H2} * Z_2^{-1} \end{bmatrix} \end{array}.$$

This method requires communication only within the calculation of the 12 and 21 subscript portion of each matrix. Thus, the more sparse those portions are, the better performance TAO can achieve. With our DOPF application, the only non-zero 12 or 21 matrix subscript is $\nabla H(x)^T$, where the only entries are based on ramping constraints. Therefore, those sections are extremely sparse (Fig. 5).

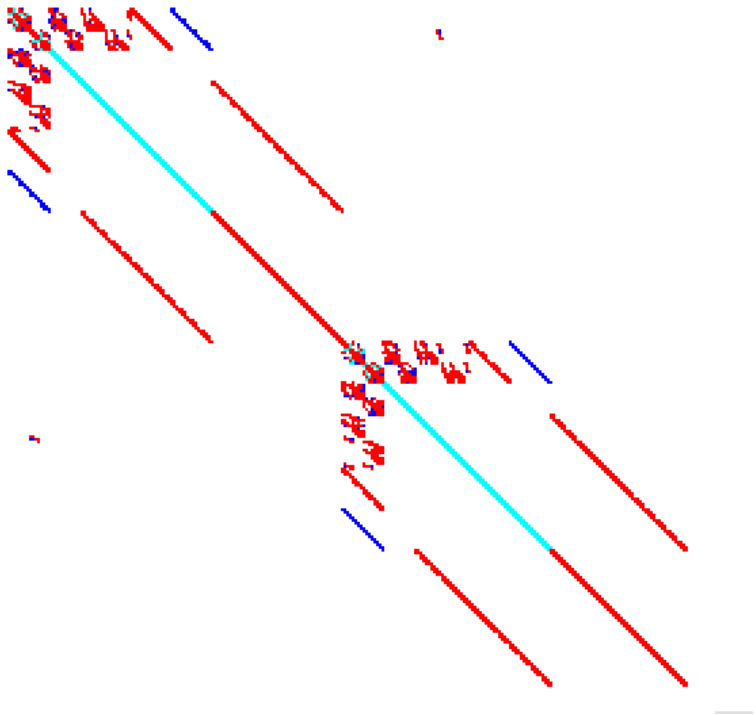

Figure 5: The nonzero distribution of the KKT matrix on two processors.

4.2.2 CONDITION NUMBER AND INERTIA

The KKT matrix (18) will become more indefinite and ill-conditioned for large-scale problems. The sequential PDIPM software packages (e.g., IPOPT and MIPS) use a reduced KKT matrix to improve computational efficiency. However, evaluation of the reduced KKT matrix requires parallel matrix-by-matrix products that incur significant data movement between processors and lead to a denser submatrix W_{xx}. Thus, we use the uncompressed KKT matrix (18) in our current PDIPM/PETSc implementation.

In the algorithm, the search direction ΔX will be guaranteed to be a descent direction (reduce $f(x)$), if the Hessian matrix W_{xx} is positive definite. To this end, we add shifts to the matrix K as necessary to guarantee it is positive definite at each linear iteration. We apply an LDL^T matrix factorization preconditioner, provided by the MUltifrontal Massively Parallel sparse direct Solver (MUMPS) library [7], to K and evaluate its inertia. To ensure a descent direction, we expect no zero inertia indices and the

number of primal and dual variables to match the number of positive and negative inertia indices, respectively. Should there be a different distribution of the matrix inertia, we introduce a shift, δ_w, to balance positive and negative inertia indices and add a shift, δ_c, to remove zero indices. This formulation can be represented as

$$\begin{bmatrix} W_{xx} + \delta_w * I & \nabla G(x)^T & -\nabla H(x)^T & 0 \\ \nabla G(x) & -\delta_c * I & 0 & 0 \\ -\nabla H(x) & 0 & -\delta_c * I & I \\ 0 & 0 & I & \Lambda_H * Z^{-1} \end{bmatrix}. \tag{26}$$

In our DOPF problem, we found that the introduced shift reduced overall run time significantly, though when a shift is applied a new factorization is required before each solve.

4.2.3 LINKING TO TAO: PDIPM

PDIPM requires the user to define several functions. These functions are: the upper and lower bounds on x, equality functions, inequality functions, the Jacobian of both equality and inequality functions, the Hessian of the objective function, and the Hessian of the equality and inequality functions. These are all defined in our DOPF code and registered with TAO using the following commands:

```
/* Objective and gradient */
ierr = TaoSetObjectiveAndGradientRoutine(tao->nlp,
      TCOPFLOWObjectiveandGradientFunction_TAO,(void*)tcopflow);
/* Equality Constraints */
ierr = TaoSetEqualityConstraintsRoutine(tao->nlp,tcopflow->Ge,
      TCOPFLOWEqualityConstraintsFunction_TAO,(void*)tcopflow);
/* Inequality Constraints */
ierr = TaoSetInequalityConstraintsRoutine(tao->nlp,tcopflow->Gi,
```

```
      TCOPFLOWInequalityConstraintsFunction_TAO,(void*)tcopflow);
/* Equality Jacobian */
ierr = TaoSetJacobianEqualityRoutine(tao->nlp,tcopflow->Jac_Ge,
      tcopflow->Jac_Ge,TCOPFLOWEqualityConstraintsJacobian_TAO,
      (void*)tcopflow);
/* Inequality Jacobian */
ierr = TaoSetJacobianInequalityRoutine(tao->nlp,tcopflow->Jac_Gi,
      tcopflow->Jac_Gi,TCOPFLOWInequalityConstraintsJacobian_TAO,
      (void*)tcopflow);
/* Set Hessian routine */
ierr = TaoSetHessianRoutine(tao->nlp,tcopflow->Hes,tcopflow->Hes,
      TCOPFLOWHessian_TAO,(void*)tcopflow).
```

These functions all require slightly different input parameters for TAO and can be easily located in the PETSc/TAO manual. Within TAO, these functions are called and updated each iteration as X updates. The only PDIPM specific difference in these routines involves the Hessian routine. For example, our Hessian routine is defined using the normal TAO format.

```
TCOPFLOWHessian_TAO(Tao nlp,Vec X, Mat H, Mat H_pre, void *ctx)
```

The void ctx variable is the type-cast tcopflow object indicated when TAO routine was set. The unique aspect comes from the W_{xx} term in the KKT matrix. The Hessian routine that PDIPM requires needs the input to be the sum of the Hessian matrices of the objective, equality, and inequality functions, which is

$$W_{xx} = \nabla^2 f(x) + \nabla^2 G(x)^T \lambda_G - \nabla^2 H(x)^T \lambda_H. \tag{27}$$

Thus, to get the dual variables the TAO function,

```
TaoGetDualVariables(Tao tao, Vec *DE, Vec *DI),
```

should be called within the Hessian routine and the vectors DE and DI are updated within TAO each iteration.

4.3 BUILDING UPON EXAGO

The Exascale Grid Optimization toolkit (ExaGO) [2] from Pacific Northwest National Laboratory is a parallel repository specifically designed to solve large-scale power grid optimisation problems. Within ExaGO there are routines for solving the power balance equations (4)-(5) tied to their pflow object, the ACOPF problem (Sec. 2.1) within opflow, and a serial DOPF problem within tcopflow solved with IPOPT. This work expanded the ExaGO project by linking it to the PDIPM solver for both solving the ACOPF problem and DOPF problem. Furthermore, the IPOPT solver the current version ExaGO uses is a serial optimization solver, therefore applying PDIPM allowed for the parallelization of this system. A comparison between these solvers is given in Sec. 5.

The ExaGO library is hierarchical similar to the previously mentioned PETSc, thus allowing the assimilaiton of a new parallel solver. Within the DOPF application, one of ExaGO's tcopflow objects are created and this object contains all the information for the system. In the tcopflow object is data for local and global vector sizes, all Vectors and Matrices used, an array of all opflow contexts, and other information not pertinent to changes made for parallelization. Within a sequential solve, the tcopflow object creates an array of opflow objects with dimension equal to Nt then, each opflow object creates an underlying pflow object that reads in the network data and builds a DMNetwork object to organize the data into a usable format. A visualization of the hierarchical structure can be seen in Fig. 6. From here, the opflow object calculates $nx = dim(x_t)$, $ng = dim(g(x_t))$, and $nh = dim(h(x_t))$ for that instance, and setting the underlying model for optimization.

Afterwards, the tcopflow object will calculate the total systems size using the sizes from each underlying opflow objects. Additionally, arrays for the starting points for each system are created, for example $nxi[1]$ is the starting location of the second time-step within the x vector i.e. x_2.

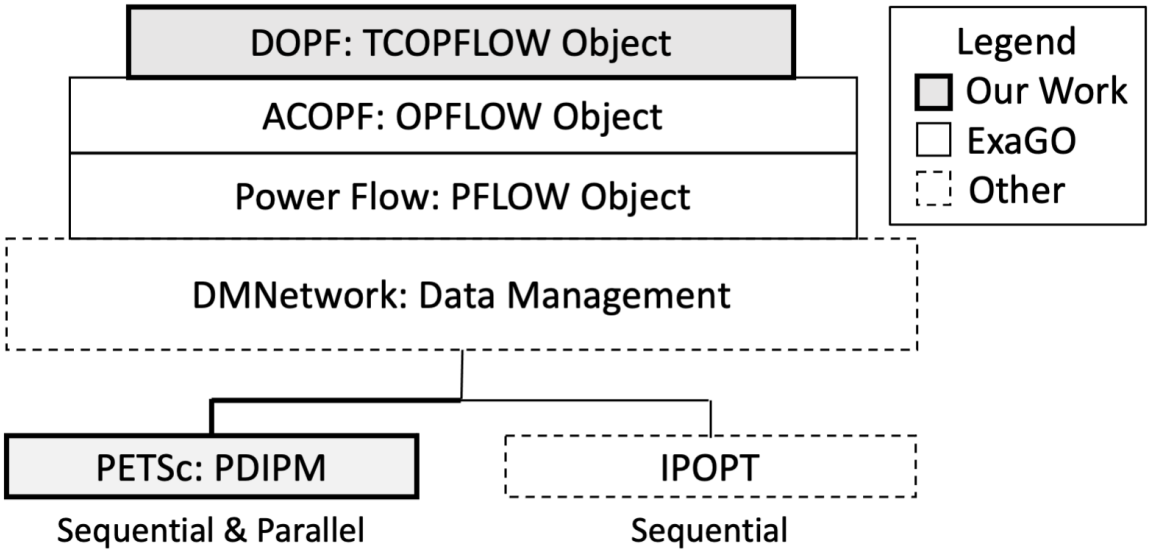

Figure 6: A visualization of the structure of ExaGO.

In parallel programming, indexing is one of the most important aspects to consider as all processors read the same instructions. The number of processors that can be utilized in the DOPF simulation must divide the total time steps Nt, creating an upper bound. This constraint allows for an easy calculation of the number of time-steps each processor contains $nt = Nt/np$ where np is the number of processors. From here, each processor builds the tcopflow object, as already described. The main distinction is that global dimensions are no longer equal to the local sizes ($Nx \neq nx$). Therefore, to create the vectors and matrices we utilize basic MPI commands MPI_Allgather and MPI_Allreduce to calculate the starting indices and global sizes.

Once the tcopflow object is populated with data, the optimization model set data is read in to set the load profiles of the network for each time-step. The load data is separated into the real and reactive load demanded at each node. Additionally, as many power grid utilize various forms of wind and solar power ExaGO supports adjusting the

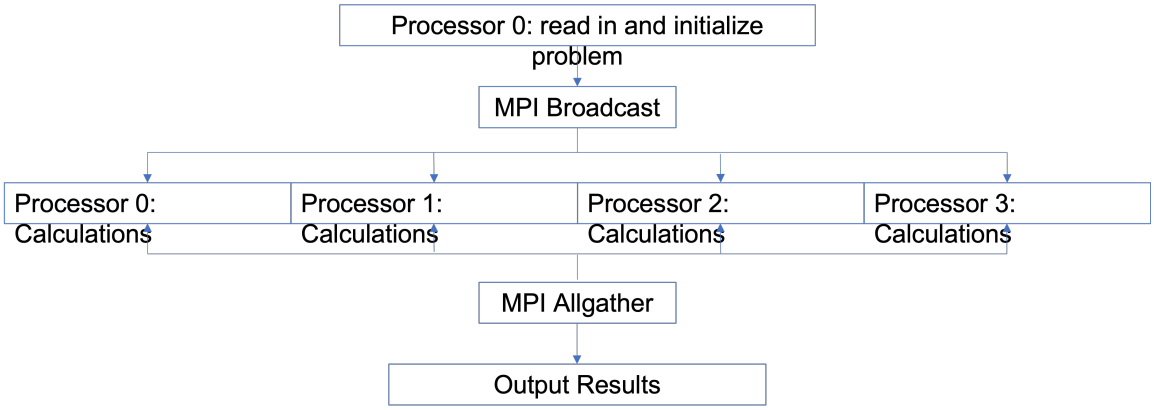

Figure 7: A visualization of parallel distribution.

power generated at these generators. For a sequential solve this data is read in for each time step and the corresponding pflow object is adjusted. When in parallel, however, each processor reads in and edits only the data for its time-steps. With each time-step updated with the desired load profiles, the optimization problem is initialized with x_0 by solving the underlying pflow system.

5 NUMERICAL RESULTS

Two standard power systems from the Texas A&M Synthetic Test Case Repository, 200-bus and 2000-bus systems [10] [19], were used to evaluate our PDIPM implementation. For DOPF results, we utilized load profiles from Jan. 2017 on the 200-bus system to demonstrate viability on real-world data [2] and two manually defined load profiles on the 2000-bus system for scope and scalability. The two load profiles for the 2000-bus system are used to demonstrate the impact of load profiles have on solution when they are within (load profile 1) or near/exceeding (load profile 2) ramping constraints on a large power network. We load data every 5 minutes for the 200-bus system and every one hour for the 2000-bus system. Thus, the number of time-steps $Nt = Duration(min)/TimeInterval(min)$.

For parallel computation, we set the number of processor cores $Np = Nt$; in other words, each core holds an independent power network subsystem at a single time-step. As discussed in Section 4, our DOPF is built on the ExaGO framework [4], which consists of ACOPF and PFLOW objects and utilizes the DMNetwork class [1] in PETSc to handle the construction and parallel distribution of the underlying power network, as shown in Fig. 6. Our DOPF code first reads network data for a single time-step as shown in Fig. 8 and 2. Using DMNetwork/PETSc API functions, we created the power network; registered network components, such as transmission lines as edges, buses and generators as vertices; added these components to the network; and then had DMNetwork assemble and distribute the resulting network to all processor cores [5]. The linear, nonlinear, and optimization solvers were built on the top of this network via standard PDIPM/PETSc API functions.

We conducted experiments on two computer systems: a Linux server and Theta, an Intel-Cray XC40 system in the Argonne Leadership Computing Facility [6]. The Linux server has dual Intel Xeon Gold 6130 CPUs at 2.1GHz with 32 cores (64 threads) and 192

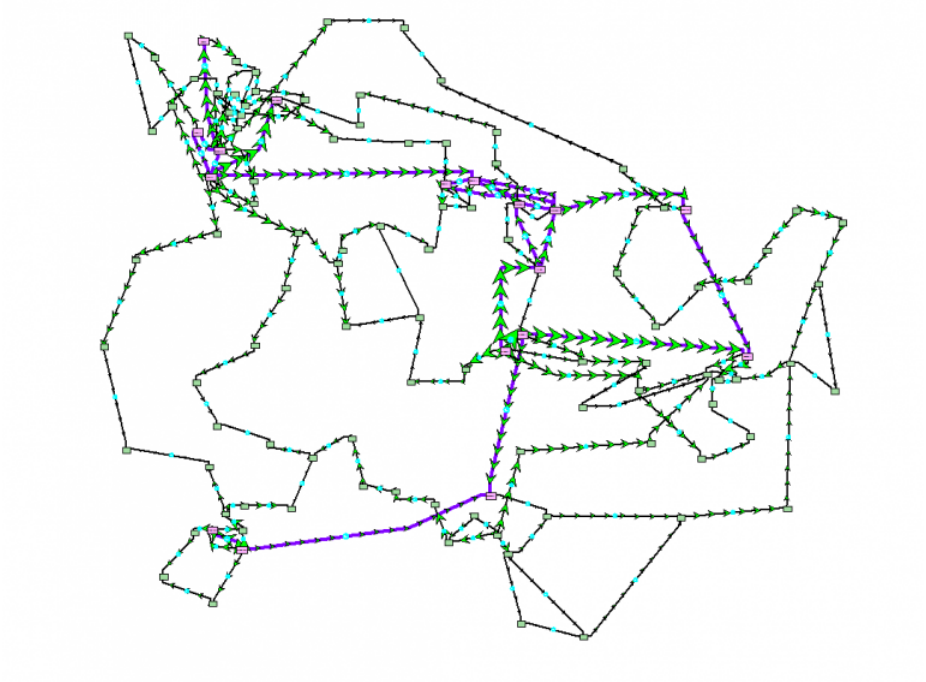

Figure 8: ACTIVSg200 network for a single time-step.
Source: [10]

GiB of RAM. Theta has 4,392 nodes, each with 64 1.3GHz Intel Xeon Phi 7230 cores with 16 GiB of MCDRAM per node.

Table 1: Total Time of 200-bus System on Theta (seconds)

Duration	Nt	NVar	$\boldsymbol{LDL}^T$		b-Jacobi	Speedup
			Np=1	Np=Nt	Np=Nt	
0.5 hr	6	25,726	216	150	56	3.9
1 hr	12	51,604	658	608	88	7.5
2 hr	24	103,360	2043	972	135	15.1
4 hr	48	206,872	6767	2704	476	14.2

NVar: Number of variables in X.
Np: Number of cores.
Speedup: Column4/Column6.

Tables 1 and 2 show the total execution time of a 200-bus system on the Theta supercomputer and on the Linux server. Tables 3 and 4 present experimental results of the 2000-Bus system utilizing the first load profile. More than 90% of the computation is spent on solving the KKT linear systems (17), for which we compare three

Table 2: Total Time of 200-bus System on the Linux Server (seconds)

Duration	Nt	IPOPT	LDL^T		b-Jacobi	Speedup
		Np=1	Np=1	Np=Nt	Np=Nt	
0.5 hr	6	13	19	15	6.3	3.0
1 hr	12	35	57	67	14	4.1
2 hr	24	102	187	108	16	11.7
4 hr	48	581	621	520	87	7.1

Np: Number of cores.
Speedup: Column4/Column6.

preconditioners used in the Generalized Minimal Residual (GMRES) Krylov subspace iterations: sequential LDL^T (Np=1), parallel LDL^T (Np=Nt), and block-Jacobi using Np diagonal blocks with sequential LDL^T applied to each inner subblock of the KKT matrix (b-Jacobi).

Table 3: Total Time of 2000-Bus System (First Load Profile) on the Linux Server (seconds)

Duration	Nt	NVar	LDL^T		b-Jacobi
			Np=1	Np=Nt	Np=Nt
2 hr	2	34,554	136	143	163
10 hr	10	172,770	Fail	693	592
20 hr	20	345,540	Fail	Fail	3,192

NVar: Number of variables in $\boldsymbol{X}$.
Np: Number of cores.
Fail: Insufficient memory during matrix factorization.

Table 4: Total Time of 2000-Bus System (First Load Profile) on Theta (seconds)

Duration	Nt	$\boldsymbol{LDL}^T$		b-Jacobi
		Np=1	Np=Nt	Np=Nt
10 hr	10	10,670	4,301	5,849
20 hr	20	Fail	Fail	20,680

Np: Number of cores.
Fail: Insufficient memory during matrix factorization.

The first load profile of the 2000-bus system (Tables: 3,4) has loosely linked inter-temporal constraints, while the second load profile produces tighter links. For the

Table 5: Total Time of 2000-Bus System (Second Load Profile) on Linux server (seconds)

Duration	Nt	$\boldsymbol{LDL}^T$		b-Jacobi
		Np=1	Np=Nt	Np=Nt
2 hr	2	251	244	1,034
10 hr	10	Out of Mem.	Out of Mem.	17,223

Np: Number of cores.
Out of Mem.: Insufficient memory during matrix factorization.

latter, the block-Jacobi preconditioner becomes less efficient, requiring far more inner linear iterations and longer execution time as shown in Table 5. We conclude that the block-Jacobi precondtioner improves parallel performance on certain systems. For large size power systems, the primary benefit is its ability to converge to the optimal solution with much less memory usage compared to the large memory overhead in $\boldsymbol{LDL}^T$ factorization.

We utilized IPOPT via ExaGO to validate the accuracy of our PDIPM. For all the test systems, our PDIPM solutions gave the numerically identical optimal values of the objective function $f(\boldsymbol{x}^*(\boldsymbol{t}))$ as IPOPT.

As a reference, Table 2 lists the numerical performance of IPOPT that is comparable with the results of LDL^T preconditioner but performs worse than our parallel PDIPM with the block-Jacobi preconditioner.

A subsystem of the 200-Bus system at a given time-step has approximately 4,287 variables, while the 2000-Bus system consists of 17,277 variables in each subsystem (i.e., $\sim$4$\times$ larger). These numbers determine the size of the diagonal blocks of the KKT matrix, as shown in Fig. 10. Although the 200-Bus system with 4-hour duration has more total numbers of variables in X than does the 2000-Bus system with 10-hour duration, its KKT matrix consists of 48 small diagonal blocks compared with 10 larger and denser diagonal blocks for the 2000-Bus system. The latter requires much larger memory for $\boldsymbol{LDL}^T$ matrix factorization.

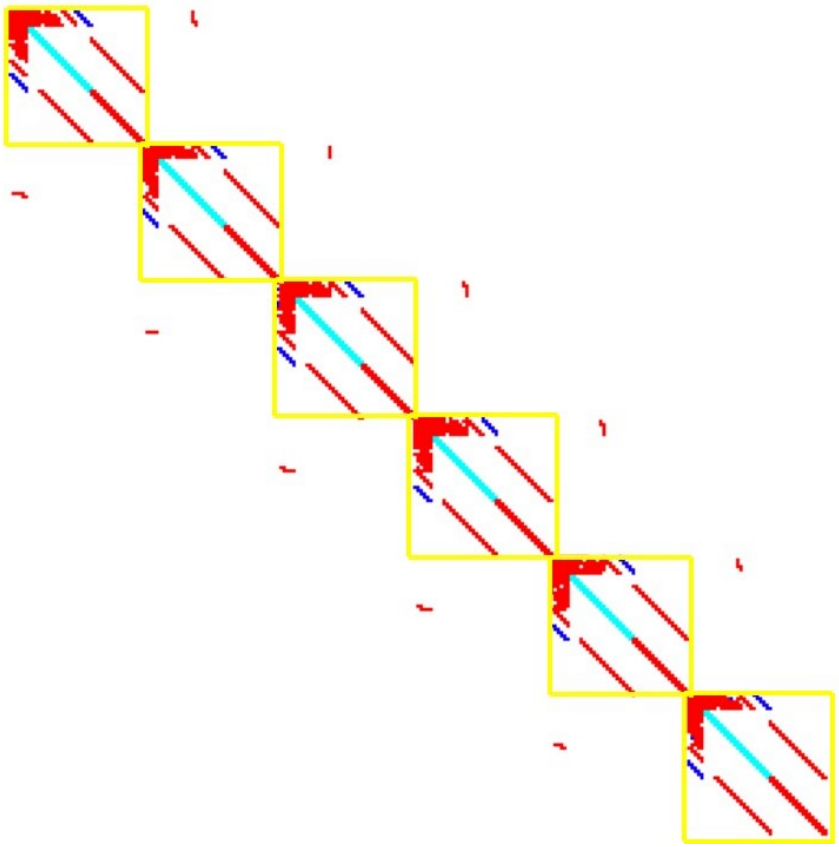

Figure 9: Visualization of the block Jacobi decomposition of the KKT matrix

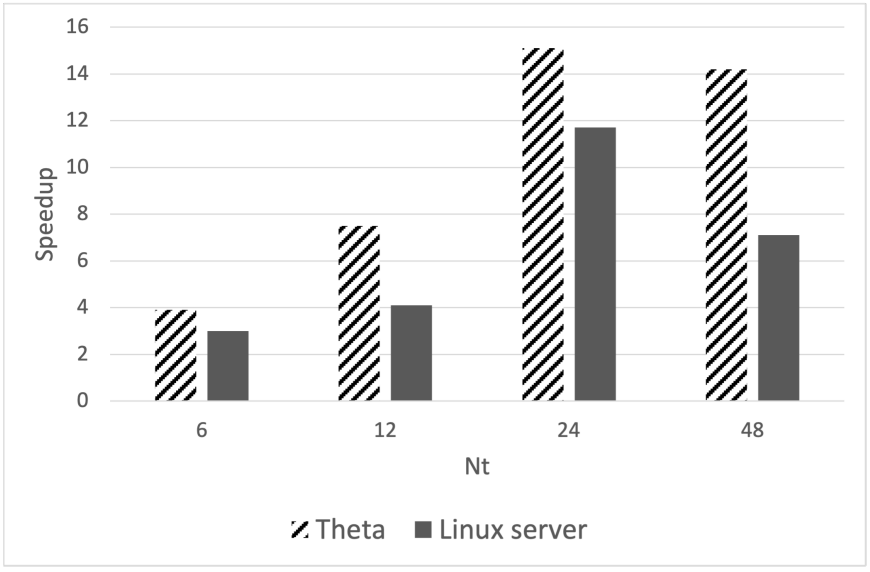

Figure 10: Speedup of block-Jacobi preconditioner on 200-Bus system on the Linux server and Theta ALCF machine.

For small-to-medium size power systems (e.g., 0.5- and 1-hour duration of 200-Bus system), parallel $\boldsymbol{LDL}^T$ does not show noticeable advantages over the sequential runs. As the size of the problems increases (e.g., 10- and 20-hour duration of 2000-Bus system), sequential runs fail due to insufficient memory during $\boldsymbol{LDL}^T$ matrix factorization, while the parallel PDIPM with appropriate preconditioners successfully computes solutions. In almost all experiments, the block-Jacobi preconditioner outperforms $\boldsymbol{LDL}^T$ and gives speedups over the sequential $\boldsymbol{LDL}^T$ ranging from 3.0 to 15.1, as shown in Fig. 9.

In addition to the solutions of the KKT systems, some researchers found the calculation of gradients and the Hessian matrices to be the next most computationally expensive aspect [20]. We calculate the analytic derivatives in the sparse gradients and Hessian matrix at each time-step simultaneously across multiple processors via ExaGO's ACOPF framework. Our experiments show that the gradient and Hessian evaluations took less than 5% of the total computation time and achieved superlinear speedup on almost all parallel tests. Table 6 presents the total time spent on the Hessian evaluations on Theta using the $\boldsymbol{LDL}^T$ preconditioner. The superlinear speedup likely comes from the cache performance. The performance of 200-Bus is significant because it would have a high cache hit ratio [17].

Table 6: Hessian Evaluations on Theta

Test System	Nt	Total Time (seconds)		Speedup
		Np=1	Np=Nt	
200-Bus	48	118	1.6	73.75
2000-Bus	10	364	16	22.8

Np: Number of cores.
Speedup: Column3/Column4.

Finally, we consider how different KSP met hods applied to the linear solve within the SNES solver Fig. 3 perform. While the standard conjugate gradient method requires a symmetric positive definite matrix, the stabilized version of the biconjugate gradient

method (BCGS) and its variations tested do not share these restrictions. Additionally, the generalized conjugate residual method (GCR) is also similar to the standard conjugate gradient method. However, it only requires that the matrix be Hermitian. However, GCR is limited by memory as it requires twice the memory of the standard GMRES solver.

Table 7: Applications of Different KSP Methods

KSP	$\boldsymbol{LDL}^T$			b-Jacobi		Speedup
Methods	Np=1	Np=12	Np=48	Np=12	Np=48	
GMRES	325.493	297.793	317.649	151.873	95.026	3.425
BCGS	371.539	306.090	355.492	110.943	65.046	5.710
FBCGSR	315.141	DNC	310.947	146.578	56.126	5.614
PipeBCGS	422.597	357.141	544.311	229.962	69.629	6.069
GCR	276.730	192.497	262.093	117.472	DNC	NA

Np: Number of cores.
Speedup: Column2/Column6.
Tests using 200-bus 4hrs. (Nt = 48)

6 CONCLUSION

This work has presented a parallel method for accelerating the multi-period dynamic optimal power flow (DOPF). We presented the methodology behind both our parallel primal dual interior point method solver, our parallel DOPF application, and their parallel implementation. We demonstrated the effectiveness of our parallel Primal Dual Interior Point Method and block-Jacobi preconditioner on the DOPF problem via scaling data on different parallel machines. In addition, this work demonstrated the speedup the parallel DOPF application achieved on different large scale power grids with varying time horizons.

REFERENCES

[1] S. Abhyankar, G. Betrie, D. Maldonado, L. McInnes, B. Smith, and H. Zhang, "PETSc DMNetwork: A library for scalable network pde-based multiphysics simulations," *ACM Transactions on Mathematical Software*, vol. 46, no. 1, 2020. DOI: 10.1145/3344587.

[2] S. Abhyankar, S. Peles, A. Mancinelli, and C. Rutherford, *Exago git repository*, https://gitlab.pnnl.gov/exasgd/frameworks/exago.

[3] S. Abhyankar, S. Peles, A. Mancinelli, and C. Rutherford, *ExaGO Manual version 1.0*, https://gitlab.pnnl.gov/exasgd/frameworks/exago/-/blob/master/docs/manual/manual.pdf.

[4] S. Abhyankar, S. Peles, A. Mancinelli, R. Rutherford, and B. Palmer, "Exascale grid optimization toolkit," 2020. [Online]. Available: https://gitlab.pnnl.gov/exasgd/frameworks/exago/-/blob/master/docs/manual/manual.pdf.

[5] S. Abhyankar, B. Smith, and E. Constantinescu, "Evaluation of Overlapping Restricted Additive Schwarz Preconditioning for Parallel Solution of Very Large Power Flow Problems," *Proceedings of the 3rd International Workshop on High Performance Computing, Networking and Analytics for the Power Grid - HiPCNA-PG 13*, 2013. DOI: 10.1145/2536780.2536784.

[6] ALCF, *Theta supercomputer*, https://www.alcf.anl.gov/support-center/theta-and-thetagpu, 2021.

[7] P. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, "Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures," *ACM Transactions on Mathematical Software*, vol. 45, 2:1–2:26, 1 2019.

[8] S. Balay, S. Abhyankar, M. F. Adams, *et al.*, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.12, 2019. [Online]. Available: https://www.mcs.anl.gov/petsc.

[9] S. Balay, S. Abhyankar, M. F. Adams, *et al.*, *PETSc Web page*, https://www.mcs.anl.gov/petsc, 2019. [Online]. Available: https://www.mcs.anl.gov/petsc.

[10] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Grid structural characteristics as validation criteria for synthetic networks," *IEEE Transactions on Power Systems*, vol. 32, no. 4, pp. 3258–3265, 2017.

[11] *Karush Kuhn Tucker conditions*. [Online]. Available: https://en.wikipedia.org/wiki/Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions.

[12] A. Mosaddegh, C. A. Canizares, and K. Bhattacharya, "Distributed Computing Approach to Solve Unbalanced Three-Phase DOPFs," *IEEE Electrical Power and Energy Conference*, 2015. DOI: 10.1109/EPEC.2015.7379985.

[13] C. G. Petra, O. Schenk, and M. Anitescu, "Real-time Stochastic Optimization of Complex Energy Systems on High-Performance Computers," *Computing in Science and Engineering*, vol. 16, no. 5, pp. 32–42, 2014, ISSN: 15219615. DOI: 10.1109/MCSE.2014.53.

[14] V. Rao, K. Kim, M. Schanen, D. A. Maldonado, C. Petra, and M. Anitescu, "A Multiperiod Optimization-Based Metric of Grid Resilience," *IEEE Power and Energy Society General Meeting*, 2019. DOI: 10.1109/PESGM40551.2019.8974137.

[15] M. Schanen, F. Gilbert, C. G. Petra, and M. Anitescu, "Toward Multiperiod AC-Based Contingency Constrained Optimal Power Flow at Large Scale," *Power System Computation Conference*, 2018. DOI: 10.23919/PSCC.2018.8442590.

[16] N. Schween, P. Gerstner, N. Meyer-HÃ¼bner, *et al.*, "A Domain Decomposition Approach to Solve Dynamic Optimal Power Flow Problems in Parallel," *Advances in Energy System Optimization*, pp. 41–64, 2018.

[17] X.-H. Sun and J. Zhu, "Performance Considerations of Shared Virtual Memory Machines," *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, pp. 1185–1194, Nov. 1995.

[18] A. Waechter and L. T. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[19] T. Xu, A. B. Birchfield, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Application of Large-Scale Synthetic Power System Models for Energy Economic Studies," *50th Hawaii International Conference on System Sciences (HICSS), Koloa, HI*, 2017.

[20] S. Zaferanlouei, H. Farahmand, V. V. Vadlamudi, and M. KorpÃ¥s, "Battpower Toolbox: Memory-Efficient and High-Performance Multi-Period AC Optimal Power Flow Solver," *IEEE TRANSACTIONS ON POWER SYSTEMS*, vol. 36, no. 5, 2021.

[21] S. Zaferanlouei, M. Korpas, J. Aghaei, H. Farahmand, and N. Hashemipour, "Computational Efficiency Assessment of Multi-Period AC Optimal Power Flow including Energy Storage Systems," *International Conference on Smart Energy Systems and Technologies*, 2018. DOI: 10.1109/SEST.2018.8495683.