LLNL Release Number: LLNL-CONF-821097

An asynchronous distributed-memory optimization solver for two-stage stochastic programming problems

Jingyi Wang
wang125@llnl.gov

Nai-Yuan Chiang
chiang7@llnl.gov

Cosmin G. Petra
petra1@llnl.gov

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California, USA

***Abstract*—We present a scalable optimization algorithm and its parallel implementation for two-stage stochastic programming problems of large-scale, particularly the security constrained optimal power flow models routinely used in electrical power grid operations. Such problems can be prohibitively expensive to solve on industrial scale with the traditional methods or in serial. The algorithm decomposes the problem into first-stage and second-stage optimization subproblems which are then scheduled asynchronously for efficient evaluation in parallel. Asynchronous evaluations are crucial in achieving good balancing and parallel efficiency because the second-stage optimization subproblems have highly varying execution times. The algorithm employs simple local second-order approximations of the second-stage optimal value functions together with exact first- and second-order derivatives for the first-stage subproblems to accelerate convergence. To reduce the number of the evaluations of computationally expensive second-stage subproblems required by line search, we devised a flexible mechanism for controlling the step size that can be tuned to improve performance for individual class of problems. The algorithm is implemented in C++ using MPI non-blocking calls to overlap computations with communication and boost parallel efficiency. Numerical experiments of the algorithm are conducted on Summit and Lassen supercomputers at Oak Ridge and Lawrence Livermore National Laboratories and scaling results show good parallel efficiency.**

***Index Terms*—SCACOPF, Optimization, Parallelization**

I. Introduction

We present a scalable algorithm and the supporting high-performance computing (HPC) implementation for solving two-stage stochastic programming problems with recourse [1]–[3]. While general to apply to various paradigms of optimization under uncertainty, the methodology presented here is driven by the problem of optimal operation of large-scale electrical transmission power grids.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The authors also acknowledge support from the U.S. Department of Energy through the Exascale Computing Program.

Electricity generation and distribution in nationwide power grid systems rely upon optimization models and tools to find the power generation injection levels and transmission power flows at each of the grid nodes such that the demand at given substations is met at the lowest generation cost and at minimum transmission losses [4]. Economic dispatch is such an optimization model routinely and extensively used by the power grid operators for intra-day scheduling operations. Recently, alternating current optimal power flow (ACOPF) models have been proposed, investigated, and in some cases adopted in operations since they model the power grid more accurately (*e.g.*, capture reactive power and include transmission losses) than the economic dispatch counterparts.

Both economic dispatch and ACOPF models are becoming increasingly challenged with the rising penetration of renewable (but highly intermittent) sources of energy (*e.g.*, wind and solar) and ongoing shifts in demand and generation, which are caused by the emergence of commodity solar systems, battery storage, and electric vehicles [5], [6]. To better accommodate these emerging technologies, the power grid operators need to operate more complex power grid systems under highly stochastic demand and generation profiles and frequent equipment failures.

Security constrained ACOPF (SCACOPF) is one of the salient emerging optimization paradigms for increasing the reliability of the power grid and ensuring its economic operation [7] under various types of failures. Used by almost every power operator worldwide, SCACOPF extends the capabilities of ACOPF by requiring that the state of the grid is secure with respect to a comprehensive list of equipment *contingencies* (*i.e.*, failures of generators, transmission lines, and transformers) and sometimes under stochastic demand and/or generation [7], [8]. As a result, the SCACOPF mathematical optimization problem reaches extreme scale as it literally needs to simulate multiple ACOPF models (as many as the number of contingencies, which are routinely $O(10^5)$) in order to find a secure state of the grid. An equally important challenge is given by the highly nonlinear and nonconvex

nature of SCACOPF (as well as of ACOPF), which makes it difficult to find global (or at least good quality) optima of the problem. On the other hand, SCACOPF models need to be solved under strict time limitations, *i.e.*, in real-time, to allow ample adjustment time for the equipment (generation ramp up or down, load shedding, transmission switching, etc.). These challenges have sparked new research over the last decades to study new scalable optimization algorithms and develop parallel computer implementations for SCACOPF problems.

One popular approach is to apply distributed optimization algorithms and solve the large system in parallel [9]. Geography-based distributed optimization algorithms where a large system is divided into smaller-scale regions interconnected through boundaries have been applied to power optimization problems and many such applications rely on HPC platforms [10]. Distributed algorithms such as the Benders Decomposition [11] that decompose the full problem into a master problem and contingency subproblems are also widely researched. Depending on the formulation of the SCACOPF, the solutions to the contingency subproblems could be used to apply additional linearized constraints to the master problem [12] or filter selected sets of contingencies [13]. Under the setup of *two-stage* stochastic programming, the master-recourse decomposition occurs naturally and the objectives from the contingencies, or *second-stage* subproblems, are part of the objective of the master problem.

The parallel computing implementation of distributed optimization algorithms has recently shown promising results for reaching real-time solutions for SCACOPF. Dandurand [14] proposed a primal-dual algorithm where the coupling constraints between the master problem and recourse subproblems are relaxed through augmented Lagrangian method. Phan [15] proposed a practical algorithm based on augmented Lagrangian constraint relaxation and the alternating direction method of multipliers (ADMM). Kraning *et al.* [16] developed a decentralized scheme based on ADMM where neighboring devices exchange messages containing solutions from the previous iteration. Linderoth *et al.* [17] introduced a decomposition algorithm for two-stage stochastic linear programming with recourse and an associated asynchronous parallel implementation. The algorithm can be roughly described as a trust-region bundle method. The optimization-based decompositions from [18] and [19], break down the SCACOPF problem at the level of the formulation into base case ACOPF and contingency response ACOPF subproblems and enforce the reconciliation between subproblems' coupling variables using first-order gradient-based methods [18] or carefully chosen approximations for the coupling terms [19]. Kim *et al.* [20] also proposed a trust-region bundle method to solve the decomposed Lagrangian master problem. They also proposed an asynchronous load scheduling algorithm to improve parallel efficiency. Other authors explored parallelization on the solver level. Qiu *et al.* [21] formed the SCACOPF problem using barrier method and developed a parallel GMRES algorithm to solve it. In [22]–[24], the SCACOPF problem is solved in parallel by decomposing the linear algebra of interior-point methods [25] using a Schur complement technique.

The present work fits under the umbrella of optimization-based decomposition and has three main contributions to existing state-of-the-art HPC for optimization. First, we propose a sequential quadratic programming approach [25] that uses exact Hessian from the base case ACOPF with Barzilai-Borwein-like spectral (gradient-based) approximations [26] for the Hessian of the contingency subproblem solutions. This could potentially lead to a fast converging (*e.g.*, higher than first-order) method while preserving the good parallel efficiency of gradient-like methods for second-stage subproblems. Second, we use a line-search-free algorithm and update the variables at each iteration based on the solution of the local subproblems, most of which can be solved in parallel. This development greatly improves computational efficiency since contingency solutions are the most computationally expensive part of the algorithm and traditional line search algorithms would require multiple such solutions each iteration. Finally, we devise a simple yet effective asynchronous scheduling strategy for reducing the load imbalance that occurs in the evaluation of the second-stage or contingency subproblems. The load imbalance is pervasive in SCACOPF and common in stochastic optimization because a handful of contingency second-stage optimization subproblems are much more impactful (and, thus, require much longer solution times) than the great majority of the subproblems.

The paper is organized as follows. In Section II, we describe mathematically the SCACOPF problem and the more abstract general two-stage stochastic programming problem. In Section III, the optimization algorithm is introduced together with necessary assumptions. We discuss in detail the adjustable update rules for the approximations of the second-stage optimal value functions. The parallel implementation of the algorithm is presented in Section IV, particularly the asynchronous Message Passing Interface (MPI) communication scheme between the base case problem and recourse subproblems. Numerical experiments on supercomputers are shown in Section V with both the convergence and scaling results.

II. Problem description and computational specific

Stochastic programming is one of the widely used paradigms of optimization under uncertainty. In this paper we consider two-stage stochastic programming problems with recourse, which can be mathematically formulated as

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) + \mathcal{R}(x) \\ \text{s.t.} \quad & c(x) = c_E \\ & d^l \le d(x) \le d^u \\ & x^l \le x \le x^u, \end{aligned} \tag{1}$$

where the so-called *recourse* function $\mathcal{R}(x) = \mathbb{E}_\Omega[r(x,\omega)]$ is defined by means of an expectation (integral) operator of the optimal value function $r(x,\omega)$ of the second-stage problem parameterized by a random vector ω over a probability space

Ω. More specifically, the second-stage optimal value function has the following mathematical form:

$$\begin{aligned} r(x,\omega) = \min_{y\in\mathbb{R}^m} \quad & p(y,x,\omega) \\ \text{s.t.} \quad & c(y,x,\omega) = c_E(\omega) \\ & d^l(\omega) \le d(y,x,\omega) \le d^u(\omega), \\ & y^l(\omega) \le y \le y^u(\omega), \end{aligned} \tag{2}$$

The second-stage problem is dependent on the *first-stage* through the coupled variable x. In (1) and (2), the functions $f(\cdot)$, $p(\cdot,\cdot,\omega)$, $c(\cdot)$, $c(\cdot,\cdot,\omega)$, $d(\cdot)$, $d(\cdot,\cdot,\omega)$ are assumed to be twice continuously differentiable and both their gradient and the Hessian are available computationally. The entries of the bound vectors d^l and $d^l(\omega)$ are in $\mathbb{R}\cup\{-\infty\}$, while the entries of the bounds vectors d^u and $d^u(\omega)$ are in $\mathbb{R}\cup\{+\infty\}$ (and the latter are strictly larger than the former). The bounds on the optimization variables x and y are such that $x^l \in (\mathbb{R}\cup\{-\infty\})^n$, $x^u \in (\mathbb{R}\cup\{+\infty\})^n$, $x^l_j < x^u_j$, for all $j \in \{1,...,n\}$, $y^l(\omega) \in (\mathbb{R}\cup\{-\infty\})^m$, $y^u(\omega) \in (\mathbb{R}\cup\{+\infty\})^m$ and $y^l_j(\omega) < y^u_j(\omega)$, for all $j \in \{1,...,m\}$ and $\omega \in \Omega$.

SCACOPF models fits under the umbrella of (1)–(2). The probability space consists of the set of all possible K contingencies, each taken with equal probability $\frac{1}{K}$. The first-stage optimization variables x in (1) correspond to power generation and power flow levels that are to be implemented instantly in practice; while the second stage variables y are recourse actions to be implemented should a contingency ω occur. We also remark that (1)–(2) finds a secure state of the power grid that minimizes current operation cost $f(x)$ plus the average monetary penalties $p(x,y,\omega)$ associated with not satisfying power demand and violating grid's power flows over all contingencies $\omega \in \Omega$.

When K is relatively small, the problem can be solved through off-the-shelf optimization packages, however, it is usually difficult to satisfy the requirement of real-time solution time. If the number of contingencies K is exceedingly large, which is common in real-world power grid operations, then solution through serial optimization solvers is intractable. We approach such problems through the decomposition of the recourse term in (1) and the use of parallel memory-distributed computing.

We remark that our optimization methodology approach is tailored to one important computational characteristic of SCACOPF, namely, the evaluation of the recourse function $r(x,\omega)$ for a given x and ω is of *considerable computational cost* as it requires solving the second-stage optimization subproblem and can reach $O(10^2)$ seconds for real-world power grids. This characteristic justifies the use of distributed computing instead of shared memory programming paradigms since the inter-node communication overhead contributes to a negligible fraction of total execution time. Further, the algorithmic choices of Section III are steered towards using as much second-order derivatives as possible without affecting the decomposition properties of the algorithm and towards a step length computation that is line search free; these choices are made specifically to reduce the number of the expensive recourse evaluations.

III. Optimization Algorithm

Various sampling strategies are usually employed to formulate (1) as a so-called sample average approximation problem (for example, see [1]). To simplify the notation, we describe the case with equal probability and recast problem (1), referred to in the remaining of the manuscript as the master problem, as

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) + \frac{1}{K}\sum_{i=1}^{K} r_i(x) \\ \text{s.t.} \quad & c(x) = c_E \\ & d^l \le d(x) \le d^u \\ & x^l \le x \le x^u. \end{aligned} \tag{3}$$

Here $r_i : \mathbb{R}^n \to \mathbb{R}$, for all $i \in 1,2,...K$, the recourse functions, are the optimal solution functions to recourse subproblems in (4), namely,

$$\begin{aligned} r_i(x) = \min_{y_i\in\mathbb{R}^m} \quad & p_i(x,y_i) \\ \text{s.t.} \quad & c(x,y_i) = c_{E,i} \\ & d^l_i \le d(x,y_i) \le d^u_i \\ & y^l_i \le y_i \le y^u_i. \end{aligned} \tag{4}$$

The problem (3) without the average of recourse terms is referred to as the *base case* problem. In many cases, $r_i(x)$ is not available analytically since the evaluation of each $r_i(x)$ for a given x requires solving an optimization problem numerically. It is essential to approximate the recourse functions $r_i(x)$ accurately throughout the solution algorithm. Similar to the sequential quadratic programming (SQP) method [25], we use a local quadratic approximation function to replace $\frac{1}{K}\sum_{i=1}^{K} r_i(x)$ in the master problem and update the approximation throughout the iterations. In this work, we assume $r_i(x)$ is at least continuously differentiable, and its function value and first-order gradient can be obtained computationally. This allows us to approximate $\frac{1}{K}\sum_{i=1}^{K} r_i(x)$ in (3) locally at $x = x_k$, k denoting the kth step in the iterative solution process, with a quadratic function $r^k(x)$. The function $r^k(x)$ takes the form of

$$\begin{aligned} & r^k(x) = r^k_0 + (g^k_0)^T(x - x_k) + \frac{1}{2}\alpha_k \left\|x - x_k\right\|^2 \\ & r^k(x_k) = r^k_0 \\ & \frac{dr^k}{dx}(x_k) = g^k_0, \end{aligned} \tag{5}$$

where r^k_0, g^k_0 denote the assembled average recourse function value and gradient, respectively, and $\alpha_k > 0$ is the quadratic coefficient. We point out that the exact second-order derivatives (assuming that they exist mathematically, which is not necessarily the case since optimal value functions $r_i(x)$ are not twice differentiable in general [27]) are matrices with considerably large dense blocks. As a result, the use of exact second-order derivatives in (5) would cause the Hessian of the

objective of the master problem (3) to become quite dense and result in a drastic increase in the computational cost associated with solving the master problem. This would have negative repercussions both on the solution time and on the parallel efficiency of the decomposition scheme. Instead we use a proximal second-order term $\frac{1}{2}\alpha_k \|x - x_k\|^2$ in (5) with a diagonal Hessian, which does not affect the sparsity pattern of the Hessian of the master problem and, thus, does not increase the computational cost of solving the master problem.

The choice of α_k is critical and problem dependent, as it is a trade-off between robust convergence behavior (large α_k) and fast but potentially unstable convergence (small α_k). Our default option is inspired from the Barzilai-Borwein (BB) gradient method [26], which can be interpreted as a secant approximation, namely, the update rule for α_k is

$$\alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}, \tag{6}$$

where $s_{k-1} = x_k - x_{k-1}, y_{k-1} = g_0^k - g_0^{k-1}$. This choice of α_k keeps the simple and sparse Hessian structure and can in practice increase the convergence rate if the recourse functions $r_i(x)$ have more favorable properties than being continuously differentiable [28].

In other cases, particularly when $\frac{1}{K}\sum_{i=1}^K r_i(x)$ is not twice continuously differentiable and highly non-linear, α_k can be viewed similar to the inverse of a trust-region radius. The larger α_k is, the smaller the step size will be. Hence, we can also update α_k based on how accurate the previous recourse approximation is, as in trust-region methods [25]. If the true recourse objectives $\frac{1}{K}\sum_{i=1}^K r_i(x_{k+1})$ is close to the approximation from the previous step $r^k(x_{k+1}) = r_0^k + (g_0^k)^T(x_{k+1} - x_k) + \frac{1}{2}\alpha_k \|x_{k+1} - x_k\|^2$, α_k can be decreased to encourage larger step size. On the other hand, if the difference between recourse approximation at the kth step and its true value evaluated at $(k+1)$th step is deemed too large, α_k should be increased and the step x_{k+1} can be rejected altogether. In all cases, problem specific α_{max} and α_{min} can be assigned to make the algorithm more efficient and robust. This is the area of the algorithm that is rich for experimentation.

Algorithm 1 Optimization algorithm with quadratic recourse approximation

Solve the base case problem to obtain x_0
Initialize α_0, iteration count $k = 0$
while Stopping criteria not reached **do**
  Solve the recourse problems at x_k with method such as interior-point methods and obtain r_0^k
  Evaluate g_0^k and construct approximation $r^k(x)$ to the recourse functions through (5)
  Take x_{k+1} as the solution to the master problem with the approximation $f(x) + r^k(x)$ or let $x_{k+1} = x_k$
  Update α_k based on the rule of choice
  Calculate the convergence measure and increase k by 1
end while

The gradient-based stopping criteria in our implementation is $\left\|g_0^{k+1} - g_0^k - \alpha_k(x_{k+1} - x_k)\right\| \leq \epsilon$, where ϵ is the error tolerance. The left side of the inequality represents the difference between the objective gradient of the true master problem at x_{k+1} and its approximation at the kth step. Therefore, upon meeting the criteria, the KKT conditions of the kth master problem solution with the approximation $r^k(x_{k+1})$ leads to the true master problem satisfying the KKT conditions to the degree of ϵ at x_{k+1}. The functional-value-based stopping criteria, which measures how much the objective can still be reduced, is $\left|(g_0^k)^T(x_{k+k} - x_k) + \alpha_k \|x_{k+1} - x_k\|^2\right| \leq \epsilon$. While not as mathematically rigorous as the gradient-based one, it can be effective and faster in practice.

The algorithm is outlined in Algorithm 1. To summarize, we start by solving the master problem with zero recourse (base case) and use the solution to initialize. Then, at the kth iteration, the recourse optimization subproblems are evaluated at x_k and its optimal function value and gradient are obtained. A local approximation to the assembled average recourse functions is established with carefully chosen α_k and used to form the approximated master problem, which is then optimized using state-of-the-art methods such as interior-point methods. The next step x_{k+1} is taken to be the solution to the master optimization problem if sufficient progress has been made. The iteration stops when the stopping criteria is met or maximum iterations have been reached.

IV. Parallel Implementation

The step of solving recourse subproblems in Algorithm 1 is ripe for parallelization. The parallel implementation is based on MPI while the algorithm itself is written in C++. In order to implement the algorithm efficiently, the processors are divided into a *master* rank and a number of *evaluator* ranks. The master rank is responsible for the distribution and assembly of recourse subproblems as well as the solution of the master problem with recourse approximation, while the evaluator rank computes the recourse subproblem assigned to it by the master rank. The recourse subproblems, as shown in (4), can have different constraints and, consequently, different optimal objectives and computing time. Therefore, they are indexed to be tracked and organized.

To start, the base case problem is solved on the master rank and the solution is broadcasted to all ranks through MPI_Bcast, a blocking operation, to initialize x. Subsequently, at each iteration, the indexed recourse subproblems are distributed to the evaluators by the master rank in a nonblocking/asynchronous manner to improve workload balancing. The master rank maintains the list of the recourse indices yet to be assigned and posts a send of the index of the next recourse subproblem to the next available evaluator.

The evaluator first posts a receive command to obtain the recourse subproblem that it is assigned to. The problem is then solved with the optimization algorithm of choice, in our case interior-point methods. Whenever an evaluator rank completes its current recourse subproblem optimization, it posts a nonblocking send back to the master rank, which

contains the function value and gradient of its recourse function. The master rank constantly checks for such a send and once it receives the signal, retrieves all relevant information. If there are more recourse subproblems left to be solved, the master sends the next index in line to the now free evaluator, and moves on to examine the other ranks. This process keeps the evaluator ranks as busy as possible to balance the different recourse subproblem load dynamically. This communication is implemented with MPI_Isend and MPI_Irecv while MPI_Requests are used to verify whether a send/receive operation is completed.

When the indices are exhausted, the master rank continues to retrieve recourse function information until the last evaluator rank has finished its task and sent back the result. It maintains a list of evaluators that have completed their jobs and moves on to the next phase in the iteration once the list contains every evaluator rank. The master rank then sends out an end signal (index number -1) to all evaluators. When an evaluator receives an end signal, it continues onto the next phase in the iteration itself as well.

The recourse functions are assembled on the master rank given the function value and gradient from individual recourse subproblems. The master problem with the recourse approximation given in (5) is solved using interior-point methods. This step is done on the master rank alone with the evaluators idle and is therefore designed to be as short as possible. The master rank then checks the convergence measure using the new solution to determine whether the program has successfully found an optimum or should enter a new iteration. The updated solution x is broadcasted (MPI_Bcast) at the end of the iteration to all ranks.

Both the master and recourse optimization problems are solved with the HiOp optimization solver [29], developed at Lawrence Livermore National Laboratory (https://github.com/LLNL/hiop). HiOp is a suite of optimization solvers for solving nonlinear programming problems of various computational specifics. It is a lightweight HPC library that leverages existing data or task parallelism to parallelize the optimization iterations. HiOp also implements specialized linear algebra kernels to achieve fine grain parallelism. For example, many of HiOp's solvers are capable of multiple computing modes including one that utilizes GPUs. The comparison of CPU computing mode and CPU-GPU hybrid mode is provided in the numerical experiments in Section V. The C++ implementation of the algorithm can be found in the pridecomp-dev branch of HiOp.

The recourse subproblems are solved using HiOp's mixed dense-sparse solver that splits the problem's matrices into sparse and dense part in a way that allows efficient use of GPUs in the linear algebra computations required by the underlying interior-point optimization algorithm [30]. The mixed dense-sparse HiOp solver is ported to GPUs using Umpire and RAJA portability libraries [31], [32] and Magma library [33], [34]. We used the CPU-GPU hybrid mode of this solver. Under this mode, the solver keeps many of its data structures (*e.g.*, vectors, matrices, linear systems, problems' derivatives, etc.) on the GPU device to reduce CPU-GPU traffic and performs the great majority of the arithmetic operations on the device. Each MPI process uses one GPU and one CPU core, the latter being primarily involved in logic control, some problem preprocessing, and interprocess communication. Currently, the interprocess (MPI-based) communication is a two-step process as it performs a CPU-GPU transfer and then the MPI transfer using the CPU memory space. We plan to revisit this communication paradigm and take advantage of the latest MPI directives that allow us to perform the interprocess transfer directly from the devices' memory spaces.

The master problem is solved using HiOp's sparse solver that is based on RAJA kernels and sparse linear solvers to perform the computations of the interior-point optimization algorithm. The porting of the HiOp's sparse solver to GPUs is currently under development and in the numerical experiments of Section V we have only used it in the CPU mode.

V. Numerical experiments

In this section we present the performance of the algorithm on supercomputing platforms. To validate and benchmark the algorithm and the distributed MPI communication scheme, numerical experiments are carried out with realistic synthetic optimization problems in the form of (3) and (4) mimicking the structure and computational specific of SCACOPF problems. In addition, we present the preliminary result of a 200-bus SCACOPF problem [35] with 50 contingencies. The algorithm applied uses the trust-region type update rule for α_k, and the objectives and constraints satisfy the assumption described in Section III. The difference in solution time for individual recourse subproblems is investigated as well.

The two supercomputers we worked on are the Lassen system at Lawrence Livermore National Laboratory and Summit at Oak Ridge National Laboratory. The Lassen system uses IBM Power9 CPUs with 44 cores and 4 NVIDIA V100 (Volta) GPUs per compute node. The CPU memory is 256 GB per node and 64 GB for GPUs. The GPU-GPU and CPU-GPU shared memory is interconnected by NVLINK 2.0. The Summit system is comprised of the IBM Power System AC922 node. Each compute node offers two IBM 22-core Power9 CPUs and six NVIDIA Tesla V100 accelerators, connected via dual NVLINK bricks. Most Summit nodes contain 512 GB of DDR4 memory for use by the POWER9 processors, 96 GB of High Bandwidth Memory (HBM2) for use by the accelerators.

First, the synthetic problem with the dimension of x at $n = 1000$ and $K = 1920$ recourse subproblems is run on Lassen from 120 MPI ranks to 960 ranks, with the recourse subproblems solved via both CPU mode and CPU-GPU hybrid mode with HiOp. For the CPU mode, we used 40 MPI processors for each compute node while CPU-GPU hybrid mode allows 4 processes per compute node due to the number of GPUs available. The results are shown in Figure 1 and 2. Both modes display high parallel efficiencies given the unbalanced nature of the recourse subproblem loads. For CPU mode, the strong scaling efficiency is 62% while for the hybrid mode it is 73.1%. It is noticeable that the CPU mode consumes less

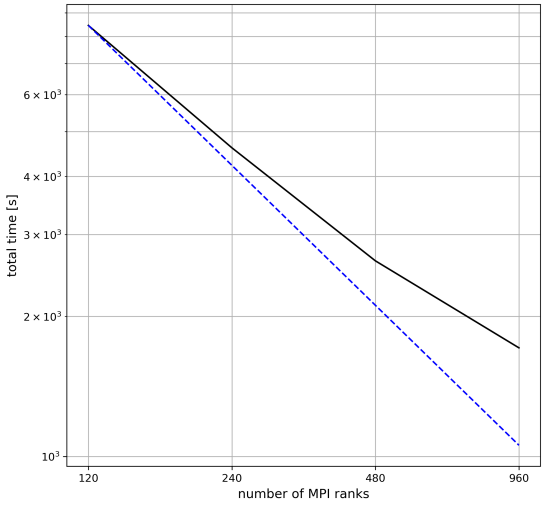

Fig. 1: Strong scaling plot on Lassen with HiOp on CPU mode.

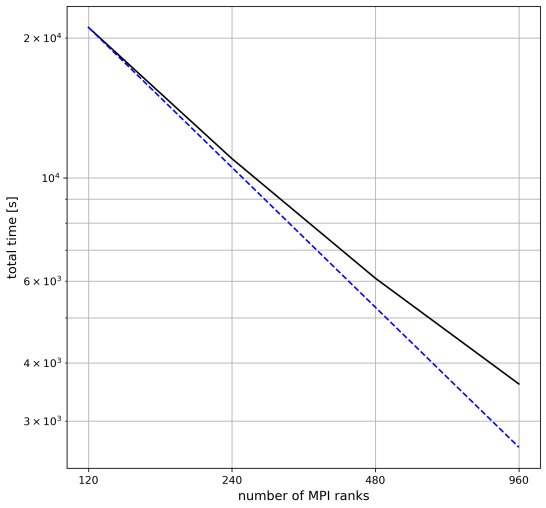

Fig. 2: Strong scaling plot on Lassen with HiOp on CPU-GPU hybrid mode.

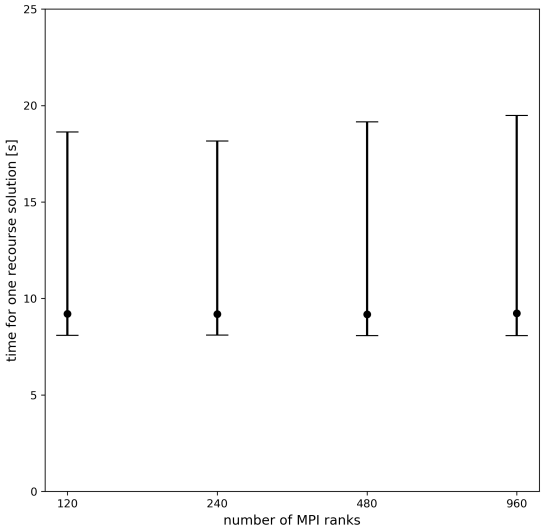

Fig. 3: Average, maximum and minimum computing time for one recourse subproblem solution, evaluated across all recourse subproblems and all iterations.

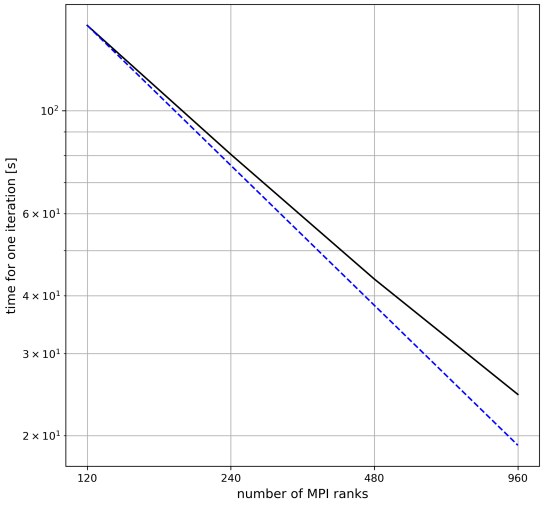

Fig. 4: Strong scaling plot on Summit for a single iteration. Dotted lines show ideal speedups.

time compared to the CPU-GPU hybrid mode overall. This leaves the parallelized part of the optimization algorithm in CPU mode a smaller percentage of the overall execution time, thus reducing scaling efficiency. The faster speed observed with the CPU mode of HiOp is caused by the fact that the GPU code is not fully optimized and that the recourse subproblems are not large enough to reach the full potential of the high-end Volta GPUs present on the two platforms we used.

Next, the same problem is run on Summit with recourse subproblems solved in CPU-GPU hybrid mode. Each compute node is assigned 6 MPI processes given its capacity of 6 GPUs. To visualize the recourse subproblem loads, we recorded the average time of solving them across all the indices and iterations on all four different number of ranks on Summit. Figure 3 illustrates the average, maximum and minimum time for each recourse subproblem. It is clear that while the average time does not change much against the number of processes, individual recourse subproblem requires drastically different amount of computing time. The maximum time for a recourse subproblem is 100% more than the minimum one. Similar behavior is observed when decomposed algorithms are applied to SCACOPF problems as well [19]. The dynamic asynchronous MPI communication scheme proposed is necessary for an efficient algorithm given such loading conditions.

The strong scaling plots of a single iteration (iteration 50) and the entire program on Summit are shown in Figure 4 and 5. High parallel efficiency can be observed on Summit, illustrating the effectiveness of the asynchronous MPI scheme. The individual iteration scaling pattern matches that of the entire program, indicating a stable parallel performance throughout iterations. The strong scaling efficiencies based on 120 processes are 77.7% and 77.4% for iteration 50 and the entire program, respectively.

On Summit, we also performed a weak scaling study,

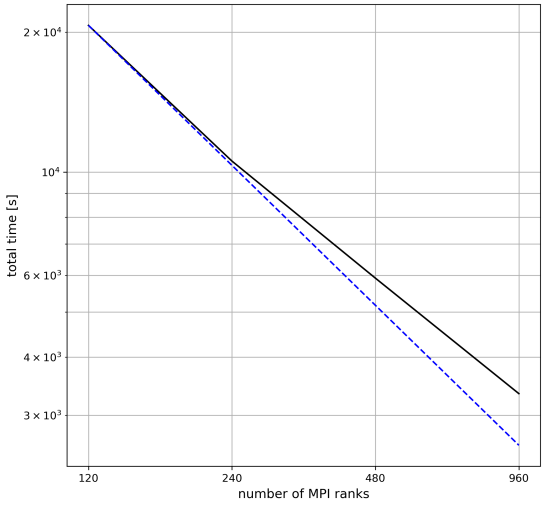

Fig. 5: Strong scaling plot on Summit for the entire problem. Dotted lines show ideal speedups.

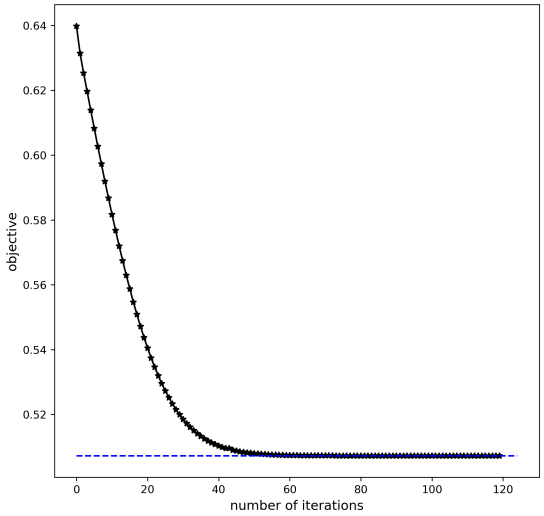

Fig. 7: Objective *v.s.* iteration for the proposed algorithm. The solution given by IPOPT is 0.50725508, marked by the blue dotted line.

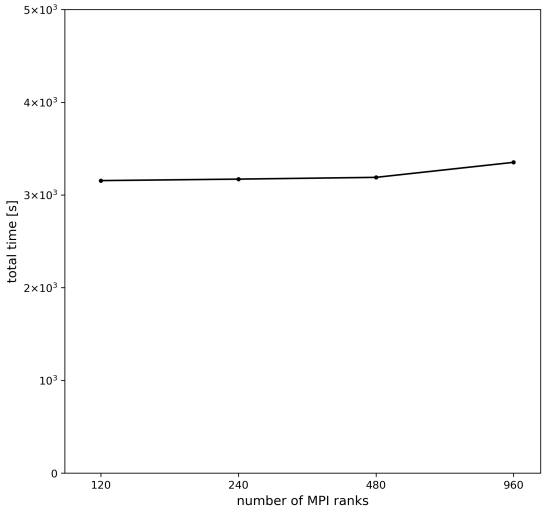

Fig. 6: Weak scaling plot on Summit.

where both the number of recourse subproblems and number of compute nodes ($n = 1000$) are varied. Each evaluator rank has two recourse subproblems per iteration to solve and the algorithm exits with the same number of iterations. The numbers of ranks used are 120, 240, 480, and 960. The result is shown in Figure 6 and the corresponding parallel efficiency, using the 120-rank case as the reference, is 94.1%. It is noted that such high efficiency is a result of the identical number of iterations and an even number of tasks per rank and should be regarded as the upper limit of weak scaling efficiency.

The algorithm result is validated through comparison with solution to the non-decomposed problem where x and y_i in (3) and (4) are all treated as independent optimization variables. The dimension of the optimization variables x and y_i are 1000 with $K = 1920$ recourse subproblems. This leads to the problem having 1.921 million variables and 1.92 million constraints. It is solved directly via IPOPT in serial. The optimal objective value is 0.507255, achieved in 51 iterations. The iteration result of the proposed algorithm is presented in Figure 7. The difference of the solutions between the two methods is in the order of 10^{-6}. The convergence rate appears linear for this problem, especially approaching convergence. Nevertheless, the algorithm is shown to successfully converge to the optimal solution within 120 iterations with functional value stopping criterion and a tolerance in the order of 10^{-7}.

Finally, a 200-bus SCACOPF problem [35] with 50 contingencies is run in HiOp GPU mode with the proposed algorithm on Lassen. In particular, this power grid network is capable of producing near-zero contingency cost which leads to a minimal number of iterations upon convergence. Due to its relatively small size, fewer ranks are used in the scaling plot shown in Figure 8. The scaling efficiency is slightly worse due to the reduced number of contingencies and iterations needed to converge.

VI. Conclusions and Future Work

We presented a scalable decomposed optimization algorithm for two-stage stochastic problems, particularly for SCACOPF problems. The algorithm constructs approximations of the recourse functions and uses an iterative numerical optimization scheme similar to sequential quadratic programming to converge to an optimal solution. The parallelization of the algorithm comes natural where the recourse subproblems are distributed asynchronously through MPI. The algorithm is shown to be convergent on numerical experiments and display high scaling efficiency on two supercomputing systems.

In the future, the algorithm will be applied to more real-world SCACOPF problems. Work will focus on improving the algorithm, particularly the quadratic coefficient to potentially converge faster. In addition, we will also apply and tune the algorithm to more complex probability distributions (*e.g.*, stochastic demand and generation in addition to contingencies) in the stochastic programming problem formulation.

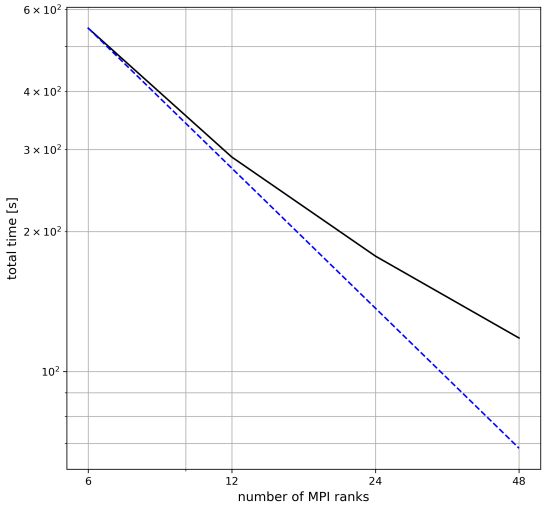

Fig. 8: Strong scaling plot on Lassen for a 200-bus SCACOPF problem with 50 contingencies. Dotted lines show ideal speedups.

REFERENCES

[1] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory, Second Edition*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014.

[2] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*. New York,: Springer-Verlag, 1997.

[3] P. Kall and S. W. Wallace, *Stochastic Programming*. Chichester: John Wiley & Sons, 2nd ed., 1994.

[4] 2017 IRC Markets Committee, "Market design executive summary," 2017. Available at: https://isorto.org/wp-content/uploads/2017/09/20170905_2017IRCMarketsCommitteeExecutiveSummaryFinal.pdf.

[5] F. Capitanescu, "Critical review of recent advances and further developments needed in ac optimal power flow," *Electric Power Systems Research*, vol. 136, pp. 57 – 68, 2016.

[6] D. K. Molzahn and I. A. Hiskens, *A Survey of Relaxations and Approximations of the Power Flow Equations*. 2019.

[7] North America Electric Reliability Corporation (NERC), "Reliability standards for the bulk electric systems of north america," July 2020. Available at: https://www.nerc.com/pa/Stand/Pages/default.aspx.

[8] S. Ekisheva and H. Gugel, "North American AC circuit outage rates and durations in assessment of transmission system reliability and availability," in *2015 IEEE Power Energy Society General Meeting*, pp. 1–5, 2015.

[9] F. Capitanescu, J. Martinez Ramos, P. Panciatici, D. Kirschen, A. Marano Marcolini, L. Platbrood, and L. Wehenkel, "State-of-the-art, challenges, and future trends in security constrained optimal power flow," *Electric Power Systems Research*, vol. 81, no. 8, p. 1731–1741, 2011.

[10] Y. Wang, S. Wang, and L. Wu, "Distributed optimization approaches for emerging power systems operation: A review," *Electric Power Systems Research*, vol. 144, p. 127–135, 2017.

[11] A. Monticelli, M. Pereira, and S. Granville, "Security-constrained optimal power flow with post-contingency corrective rescheduling," *IEEE Transactions on Power Systems*, 2 1987.

[12] Y. Li and J. McCalley, "Decomposed scopf for improving efficiency," *IEEE Transactions on Power Systems*, vol. 24, pp. 494–495, 2009.

[13] F. Capitanescu and L. Wehenkel, "A new iterative approach to the corrective security-constrained optimal power flow problem," *IEEE Transactions on Power Systems*, vol. 23, pp. 1533 – 1541, 12 2008.

[14] B. Dandurand and K. Kim, "Scalable decomposition methods for preventive security-constrained optimal power flow," in *2018 Power Systems Computation Conference (PSCC)*, pp. 1–7, 2018.

[15] D. Phan and J. Kalagnanam, "Some efficient optimization methods for solving the security-constrained optimal power flow problem," *IEEE Transactions on Power Systems*, vol. 29, pp. 863–872, 03 2014.

[16] M. Kraning, E. Chu, J. Lavaei, and S. P. Boyd, "Dynamic network energy management via proximal message passing," *Found. Trends Optim.*, vol. 1, pp. 73–126, 2014.

[17] J. T. Linderoth and S. J. Wright, "Decomposition algorithms for stochastic programming on a computational grid," *Computational Optimization and Applications*, vol. 24, pp. 207–250, 2003.

[18] L. Liu, A. Khodaei, W. Yin, and Z. Han, "A distribute parallel approach for big data scale optimal power flow with security constraints," in *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 774–778, 2013.

[19] C. G. Petra and I. Aravena, "Solving realistic security-constrained optimal power flow problems," *Operations Research*, vol. submitted, 2021.

[20] K. Kibaek, C. Petra, and V. Zavala, "An asynchronous bundle-trust-region method for dual decomposition of stochastic mixed-integer programming," *SIAM Journal on Optimization*, vol. 29, pp. 318–342, 01 2019.

[21] W. Qiu, A. J. Flueck, and F. Tu, "A parallel algorithm for security constrained optimal power flow with an interior point method," in *IEEE Power Engineering Society General Meeting, 2005*, pp. 447–453 Vol. 1, 2005.

[22] N. Chiang, C. G. Petra, and V. M. Zavala, "Structured nonconvex optimization of large-scale energy systems using pips-nlp," in *2014 Power Systems Computation Conference*, pp. 1–7, 2014.

[23] C. G. Petra, O. Schenk, M. Lubin, and K. Gärtner, "An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization," *SIAM Journal on Scientific Computing*, vol. 36, no. 2, pp. C139–C162, 2014.

[24] 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. 99, pp. 1–9, 2014.

[25] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer, 2nd ed., 2006.

[26] J. Barzilai and J. M. Borwein, "Two-point step size gradient methods," *IMA Journal of Numerical Analysis*, vol. 8, pp. 141–148, Jan 1988.

[27] J. F. Bonnans and A. Shapiro, *Perturbation Analysis of Optimization Problems*. New York: Springer, 1 ed., 2000.

[28] M. Raydan, "On the barzilai and borwein choice of steplength for the gradient method," *IMA Journal of Numerical Analysis*, vol. 13, pp. 321–6, Jul 1993.

[29] C. G. Petra, "A memory-distributed quasi-Newton solver for nonlinear optimization with a small number of constraints," *J. of Parallel and Distributed Computing*, vol. in print, 2018.

[30] S. Peles, M. Perumalla, M. Alam, A. J. Mancinelli, R. C. Rutherford, J. Ryan, and C. G. Petra, "Porting the nonlinear optimization library hiop to accelerator-based hardware architectures," *Journal of Scientific Computing*, vol. submitted, 2021.

[31] D. A. Beckingsale, M. J. Mcfadden, J. P. Dahm, R. Pankajakshan, and R. D. Hornung, "Umpire: Application-focused management and coordination of complex hierarchical memory," *IBM Journal of Research and Development*, vol. 64, pp. 00–1, Nov. 2019.

[32] D. A. Beckingsale, R. H. J. Burmark, H. Jones, W. Killian, A. J. Kunen, P. R. O. Pearce, B. S. Ryujin, and T. R. W. Scogland, "Raja: Portable performance for large-scale scientific applications," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pp. 71–81, 2019.

[33] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Computing*, vol. 36, pp. 232–240, June 2010.

[34] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "Accelerating numerical dense linear algebra calculations with gpus," *Numerical Computations with GPUs*, pp. 1–26, 2014.

[35] 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, pp. 3258–3265, 2017.