IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 25, 2018, accepted February 28, 2018, date of publication March 5, 2018, date of current version March 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2812084

Fast Batched Solution for Real-Time Optimal Power Flow With Penetration of Renewable Energy

SHENGJUN HUANG[1,2], (Student Member, IEEE), AND VENKATA DINAVAHI[1], (Senior Member, IEEE)
[1]Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta T6G 2V4, Canada
[2]College of Information System and Management, National University of Defense Technology, Changsha 410073, China

Corresponding author: Shengjun Huang (shengjun@ualberta.ca)

This work was supported by the Natural Sciences and Engineering Research Council of Canada. The work of S. Huang was supported by the China Scholarship Council under Grant 201403170337.

ABSTRACT Renewable energy systems have become an integral part of modern power grid operation, where the forecasting error is inevitable even though advanced prediction techniques are utilized. To improve the solution efficiency and accuracy of real-time optimal power flow (RTOPF), a three-stage framework for parallel processing is employed in this paper. In Stage 1, uncertainties from renewable generators and demand loads are characterized with scenarios. Large numbers of RTOPFs corresponding to each scenario are formulated and addressed in Stage 2, where the linear systems are regulated into the same sparsity pattern and then tackled in a batched style with the graphics processing unit (GPU). Results from Stage 2 are utilized in Stage 3 to perform a hot-start RTOPF, where the forecasting error can be minimized. Case studies are implemented on the IEEE 14-bus, 57-bus, 118-bus, and 300-bus systems with 1024 scenarios. The superiority of the batched GPU solution has been validated by comparisons with regular GPU, parallel CPU, and sequential CPU implementations. Discussions on the batch size and hot-start strategy are also presented.

INDEX TERMS Graphics processing unit, parallel processing, real-time optimal power flow, renewable energy.

I. INTRODUCTION

Introduced by Carpentier in 1962 [1], Optimal Power Flow (OPF) has been widely utilized to determine the optimal settings of a power system to achieve various objectives. Due to the variation of grid status, the device control instructions should be updated timely, especially in the context of modern smart grids, where the penetration of the intermittent Renewable Energy Generators (REGs) is remarkable. Therefore, Real-Time OPF (RTOPF) [2] is desired to make fast decisions for the compensating of fluctuations. Since forecasted values of REG generation and demand load are employed as the input of RTOPF, the output decisions should be utilized when the forecasted values are still valid, i.e., the RTOPF solution process should be so fast that the difference between forecasted and realized values is minimal. Otherwise, constraints might be violated, resulting in the deterioration of RTOPF solution. In order to address this concern, two directions can be explored: increasing the prediction accuracy and accelerating the solution process. In terms of forecasting, two methods are developed in the literature:

- *Probabilistic method:* Suppose the Probability Distribution Functions (PDFs) of the uncertain demands and REGs are given, the stochastic optimization problem is formulated and addressed with Chance-Constrained Programming (CCP) technique, which ensures that each constraint will be satisfied in a predefined high probability. The CCP is a relatively robust approach, nevertheless, it is complex and difficult to solve and; therefore, its application is usually restricted to long-term off-line optimization [3]–[5].
- *Deterministic method:* Instead of performing the prediction with historical data and mathematical statistical model, different types of sensors designed for temperature, luminance, and wind speed, etc., can be widely utilized for observing. Since the obtained data set consists of fixed real values rather than PDFs, a deterministic optimization problem can be formulated, whose solution is easier and more accurate when compared with its probabilistic counterpart. The detailed solution process will be elaborated in the following paragraphs.

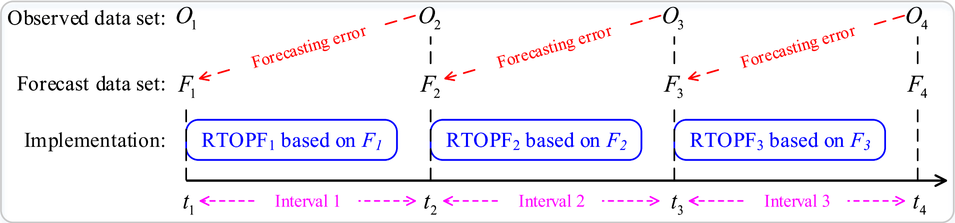


FIGURE 1. Traditional implementation framework of RTOPF.

Based on the above analysis, the deterministic method is employed for the solution of RTOPF. As reported in [6]–[8], the RTOPF is carried out in every 5–15 min intervals based on the static snapshot forecasted data. Fig. 1 illustrates the traditional implementation framework of RTOPF. At the beginning of each interval i, RTOPF$_i$ is carried out to produce the control decisions for interval $i+1$. Theoretically, the real-time observed dataset O_{i+1} should be utilized as the input for RTOPF$_i$ since the decision is made for interval $i+1$. Nevertheless, O_{i+1} cannot be gathered until t_{i+1}, i.e., utilizing O_{i+1} at t_i is not achievable. Therefore, forecasted dataset F_i is generated for the substitution of O_{i+1}. Since F_i is derived from O_i, and O_i is different from Q_{i+1} as their time-stamps are various, therefore the variation between F_i and O_{i+1} is inevitable, and it goes higher as the length of interval increases. In order to mitigate the forecasting error associated with REGs and loads, a lot of research effort has been put on the development of advanced physical and statistical algorithms, such as conditional kernel density estimation [9], cooperative control [10], artificial neural network [11]–[13], fuzzy-based methods [14], and numerical weather prediction grids [15], etc.

The aforementioned methodologies provide one promising direction to improve the accuracy of deterministic RTOPF; however, the prediction technique is approaching to its maximum capability. Therefore, further improvements from other directions have been explored. In [6], the regular RTOPF scheduling interval (10 min) has been divided into several subintervals (1 min) based on the introduction of participation factors for each generator. The time resolution has been enhanced, thus the prediction error was reduced. On the other hand, acceleration on the solution process of RTOPF is also constructive for the reduction of the length of intervals in Fig. 1. The projected gradient descent was employed for on-line OPF in [16], where good performance has been reported for both solution efficiency and global convergence. Reference [17] combines the genetic algorithm and two-point estimate method to effectively tackle the uncertainties. Quasi-Newton method is resorted to developing a real-time algorithm for AC OPF in [18], where the second order information is utilized to provide suboptimal solutions on a fast timescale.

Although these methods are valid and beneficial, they share one common limitation that F_i in Fig. 1 is directly utilized to represent O_{i+1}, i.e., there is an implicit hypothesis that the forecasted data set is 100% acceptable during each interval, which may not always be true. To address this issue, several F_i were generated to predict Q_{i+1} in [19]. By solving different RTOPFs corresponding to each F_i, a lookup table was maintained. At time t_{i+1}, the O_{i+1} is available, then the closest F_i can be determined, thus quick decisions can be made by indexing the lookup table. This solution framework is beneficial to minimize the forecasting error between F_i and O_{i+1}; however, the intensive computational burden due to large numbers of F_i consists one of its greatest drawbacks, thus only 2 REGs and 49 scenarios were considered in [19]. In addition, the difference between F_i and O_{i+1} might still be large after minimization.

To address these two concerns, the Graphics Processing Unit (GPU) is introduced in this work for the acceleration of lookup table formulation since RTOPFs corresponding to each F_i are independent and can be solved in parallel. Based on this table and the realization of O_{i+1}, the result corresponding to the closest F_i is selected and utilized as the initial point for a RTOPF based on O_{i+1}, i.e., a hot-start RTOPF is implemented. GPU stimulates the solution process of RTOPF in a batched style, whose superiority over regular CPU and GPU implementation is validated with case studies. Hot-start eliminates the variation between F_i and O_{i+1} because the last RTOPF is actually designed for O_{i+1}, and the solution time is very short due to the high quality of start point. As a High-Performance Computing (HPC) platform, GPU has been previously used for developing parallel solution algorithms for various power system studies, such as transient stability simulation [20], [21], electromagnetic transient simulation [22], [23], ionized field computation [24], and dynamic state estimation [25], [26]. In addition, GPU has been harnessed to accelerate the solution of OPF by [27], [28]; however, batched implementation is not related and the uncertainty of REGs and loads are not considered. To the best of our knowledge, utilizing GPU to improve the accuracy of the forecasting data for RTOPF with REGs has not been reported before. In addition, implementation strategies and batch sizes are investigated and discussed.

The proposed framework has several distinct features:

- It minimizes the forecasting error by the integration of GPU acceleration and hot-start strategy, which provides a potential for the solution of RTOPF with REGs.
- It scrutinizes the uncertainty of REGs and loads with scenarios, where scenario reduction algorithms can be utilized to control the size, which has a close relationship with the hot-start strategy, i.e., the larger the size, the less iteration of hot-start RTOPF solution, and vice versa.

- It can be implemented on either pure CPU or heterogeneous CPU-GPU architectures, where several details are referable for other applications, such as regulation rules for heterogeneous computing and data reusing schemes for the batched solution.
- It provides different configurations to address specified problems, including various fill-in reduction algorithms, platforms, and batch sizes.

The rest of this paper is organized as follows. Section II formulates the RTOPF problem and uncertainty. Section III develops the GPU-based parallel implementation framework of RTOPF with the batched mode. Case studies and discussions are provided in Section IV. Conclusions and future works are provided in Section V. Finally, the Primal-Dual Interior Point Method (PDIPM) is briefly reviewed in Appendix A to facilitate the description of GPU implementation in Section III.

II. MATHEMATICAL FORMULATION

This section introduces the definition and formulation of classical RTOPF, where all the utilized variables and parameters are defined in the nomenclature. To achieve the efficiency and accuracy, a three-stage process is developed for the solution of RTOPF with REGs, where scenario reduction algorithm and hot-start strategy will be employed.

A. NOMENCLATURE

$\mathcal{N}_b, \mathcal{N}_l$	Set of buses and lines.
$\mathcal{N}_g, \mathcal{N}_r$	Set of buses where thermal and renewable generators are integrated.
a_g, b_g, c_g	Coefficients of the quadratic cost function of generator g.
P_g^G, Q_g^G	Active and reactive power output of thermal generator g.
P_r^R, Q_r^R	Active and reactive power output of REG r.
P_i^D, Q_i^D	Active and reactive power demand of bus i.
G_{ij}, B_{ij}	Transfer conductance and susceptance between buses i and j.
V_i, V_j	Voltages magnitude at node bus i and j.
P_g^{G0}	Active power output of generator g in the previous subinterval.
f_{ij}	Power flow on line ij.
θ_{ij}	Voltage angle difference between buses i and j.
θ_i, θ_j	Voltage angle at node bus i and j.
$^{down}, ^{up}$	Ramp down and up limits of thermal generator.
$^{min}, ^{max}$	Lower and upper limits of specified variables.

B. OPTIMIZATION MODEL

The RTOPF is expressed as "determining the thermal generator output to minimize the total operation cost based on the forecasted power output of REGs, demand loads, and current status of thermal units".

1) OBJECTIVE FUNCTION

The OPF can be designed to optimize a wide range of goals [29], such as total generation cost, system loss, and emission, etc. The operation cost of the generator is determined for optimization in this work, which is expressed by a quadratic function as follows:

$$\min F = \sum\nolimits_{g\in\mathcal{N}_g}\left[a_g\left(P_g^G\right)^2 + b_g P_g^G + c_g\right]. \qquad (1)$$

2) NODAL POWER BALANCE CONSTRAINTS

The active and reactive power balance on bus $i \in \mathcal{N}_b$ can be written as:

$$P_i^G + P_i^R - P_i^D = V_i \sum_{j\in\mathcal{N}_b} V_j(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}), \qquad (2)$$

$$Q_i^G + Q_i^R - Q_i^D = V_i \sum_{j\in\mathcal{N}_b} V_j(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}). \qquad (3)$$

If $i \notin \mathcal{N}_g$ or $i \notin \mathcal{N}_r$, then $P_i^G = Q_i^G = 0$ and $P_i^R = Q_i^R = 0$, respectively. It should be noted that the intersection of $\mathcal{N}_g$ and $\mathcal{N}_r$ may not be empty.

3) GENERATOR CAPACITY CONSTRAINTS

For generator $g \in \mathcal{N}_g$, its active and reactive power output are limited, which can be expressed as:

$$P_g^{G,min} \le P_g^G \le P_g^{G,max}, \qquad (4)$$

$$Q_g^{G,min} \le Q_g^G \le Q_g^{G,max}. \qquad (5)$$

4) RAMP RATE CONSTRAINTS

The ramp rate limit of generator $g \in \mathcal{N}_g$ is considered as follows:

$$P_g^{G0} - R_g^{G,down} \le P_g^G \le P_g^{G0} + R_g^{G,up}. \qquad (6)$$

5) VOLTAGE DEVIATION CONSTRAINTS

The voltage magnitude at bus $i \in \mathcal{N}_b$ is restricted by its lower and upper limits:

$$V_i^{min} \le V_i \le V_i^{max}. \qquad (7)$$

6) LINE SECURITY CONSTRAINTS

The maximum power flow on line $ij \in \mathcal{N}_l$ should not exceed its capacity for security concerns:

$$f_{ij}^{min} \le f_{ij} \le f_{ij}^{max}. \qquad (8)$$

In [30], the constraints (8) have been proven to be practically equivalent to (9).

$$\theta_{ij}^{min} \le \left(\theta_{ij} = \theta_i - \theta_j\right) \le \theta_{ij}^{max}. \qquad (9)$$

C. UNCERTAINTY MANAGEMENT

It should be noted that in the above equations (2)–(3), the terms P_i^R, P_i^D, Q_i^R, and Q_i^D are fixed forecasted values of the next interval. Practically, the output of REG is greatly dependent on the meteorological data [32], such as solar irradiation and wind speed. Fig. 2 gives an example at NREL Solar Radiation Research Laboratory (latitude 39.74°N, longitude 105.18°W, elevation 1829m, and time zone GMT-7)

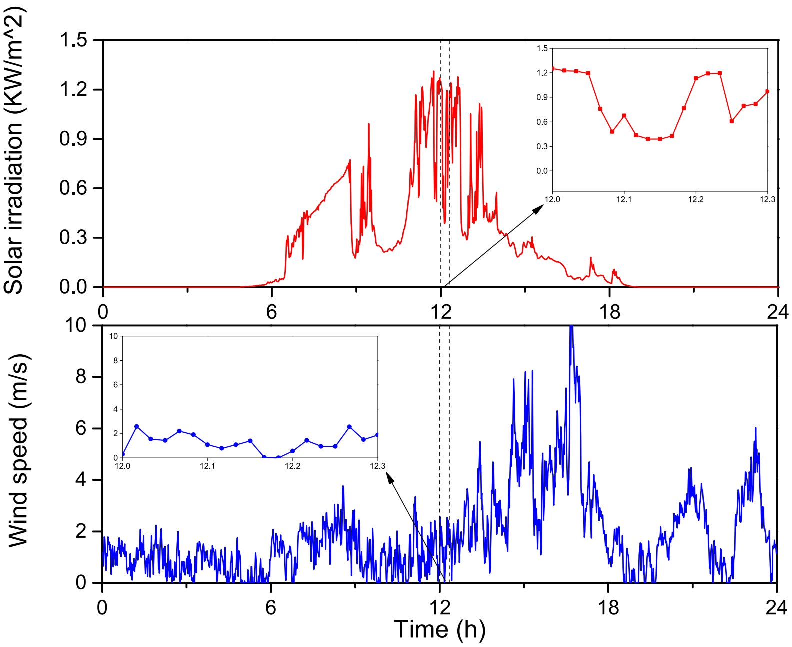


FIGURE 2. Daily solar irradiation (global CMP22) and wind speed (height 19ft) at NREL Solar Radiation Research Laboratory on May 2, 2017 [31].

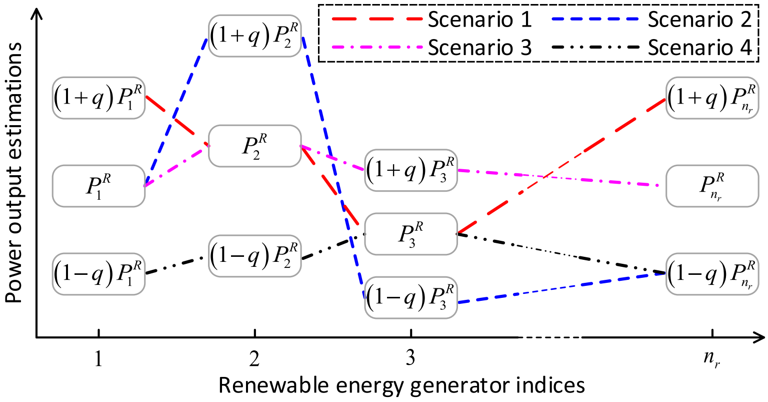
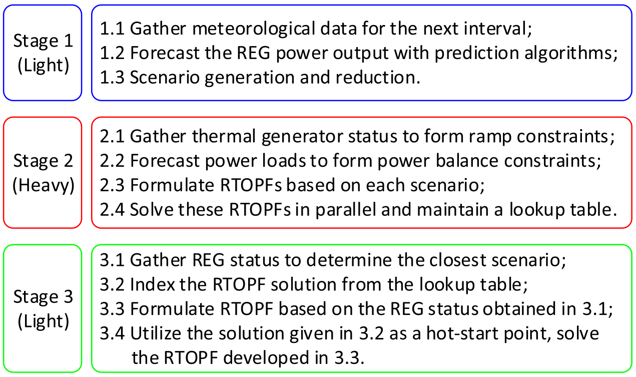


FIGURE 3. Illustration of scenarios for the power output of REGs.

on May 2, 2017 [31]. It can be seen that, for hours resolution, the fluctuation is large and random, i.e., the prediction accuracy is not guaranteed. In the sub-figure, the variation is depicted in every minute, which is more stable and easier to forecast. In this paper, although the decision interval is determined as 30s, the prediction accuracy is still limited. Therefore, additional enhancement might be required.

Consider the forecasted active power output of REG r is P_r^R, probable scenarios around P_r^R are also likely to be realized, such as $(1\pm q)P_r^R$, where q can be 5% or 10%. Take Fig. 3 as an example, picking one value of these 3 high probable estimations of each REG, a scenario can be generated. It is obvious that the total number of scenarios is 3^{n_r} if there are n_r REGs, which will increase sharply if the dynamics of loads and more values of q are considered. For each scenario, there will be one RTOPF to be built and addressed. More scenarios represent higher accuracy, but the computation may be excessive. Thus, scenario reduction strategies should be employed to strike a balance between the heavy computational burden and prediction accuracy.

Fig. 4 demonstrates the stages of the whole solution process of RTOPF with REGs integrated. Stage 1 is devoted to scenario preparation, where the utilized prediction algorithm and scenario reduction strategy are out of the scope of this paper, for more details please refer to [15], [33], and [34]. RTOPF formulation and solution is performed in Stage 2, where the heaviest computation resources are consumed. To achieve high solution efficiency, GPU-based acceleration techniques will be introduced in this paper. Stage 3 is designed to refine the solution specified from the look-up table and eliminate the forecasting error.



Stage 1 (Light)

1.1 Gather meteorological data for the next interval;
1.2 Forecast the REG power output with prediction algorithms;
1.3 Scenario generation and reduction.

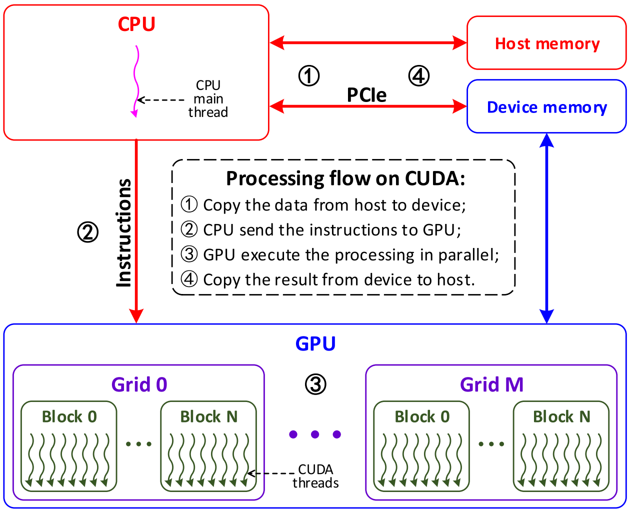
Stage 2 (Heavy)

2.1 Gather thermal generator status to form ramp constraints;
2.2 Forecast power loads to form power balance constraints;
2.3 Formulate RTOPFs based on each scenario;
2.4 Solve these RTOPFs in parallel and maintain a lookup table.

Stage 3 (Light)

3.1 Gather REG status to determine the closest scenario;
3.2 Index the RTOPF solution from the lookup table;
3.3 Formulate RTOPF based on the REG status obtained in 3.1;
3.4 Utilize the solution given in 3.2 as a hot-start point, solve the RTOPF developed in 3.3.

FIGURE 4. Three-stage solution process of the hot-start RTOPF with REGs.



Processing flow on CUDA:
① Copy the data from host to device;
② CPU send the instructions to GPU;
③ GPU execute the processing in parallel;
④ Copy the result from device to host.

FIGURE 5. CUDA thread hierarchy and processing flow.

III. PARALLEL IMPLEMENTATION ON GPU

This section summarizes general features of Nvidia® GPU architecture and heterogeneous computing, with processing flow and instruction rules are described. According to these regulations, a detailed parallel implementation scheme of PDIPM for the solution of RTOPF is elaborated based on the details given in Appendix A. Finally, the batched linear solver is presented. Other types of GPU architectures, such as AMD®, are out of the scope of this paper.

A. GPU AND COMPUTE UNIFIED DEVICE ARCHITECTURE

Initially designed for graphic and image processing, GPU has been introduced for the general-purpose scientific computing in recent years [35]. Its popularity on HPC community was soared by the release of Nvidia® Compute Unified Device Architecture (CUDA) [36]. In CUDA, the function designed for GPU implementation is named as *kernel*. Different from regular C function where one call means one execution, the *kernel* can be executed simultaneously by multiple *CUDA threads* with one call. As shown in Fig. 5, the threads are organized in a three-level hierarchy, i.e., each *thread* is contained in a *block* and many *block* consist of a *grid*. During execution, the dimensions of *grid* and *block* are specified, as well as their indices, thus each *thread* can be uniquely identified and controlled. The CUDA programming model is heterogeneous, i.e., all the *kernels* execute on GPU while the rest of the C program run on CPU. In addition, the host and device memories are logically separated. *CUDA threads* can access the device memory during execution, including local memory, shared memory, global memory, etc. *CPU thread* manages the bilateral data transfer between the host and device memory via the API function *cudaMemcpy()*, which is physically performed by PCIe interface. Based on these restrictions, a brief processing flow is given in Fig. 5.

In GPU, each *thread* is executed on one *CUDA core*, an arithmetic unit contained in the *Streaming Multiprocessor* (SM). The SM receives computation task in the unit of *block*. Each *thread* in one *block* can be concurrently executed in one SM, and one SM can also take care of multiple *blocks* at the same time. Different blocks are independent with each other, while threads within one block can cooperate via shared memory and barrier.

B. RULES FOR HETEROGENEOUS COMPUTING

Although GPU has far more cores than CPU, its frequency is significantly lower than CPU core and several important features are missing, such as interrupts and virtual memory; therefore, GPU may not always outperform CPU [37]. Actually, the GPU is productive in the manipulation of large amounts of data with many streams, while the CPU excels at doing complex operations on a small set of data.

The SM organizes threads in groups of 32 and called *warps*. If all 32 threads in a *warp* are on the same path, the execution is coherent and efficient [36]. On the other hand, if divergence occurred on the data-dependent conditional branch, each thread will be checked individually and sequentially, i.e., parallelism is limited. Similarly, the *warp* memory access share the same characteristic. If the target address of all 32 threads is successive, the coalesced memory request will be performed; otherwise, up to 32 requests may be launched. The memory access pattern consists one of the most important obstacles for the performance of GPU computing [38]. The GPU-accelerated libraries provided by Nvidia® contribute to a potential. With minimal changes, one can integrate them into the domestic code. Since these libraries are highly optimized, the performance is usually remarkable [28]. It should also be noted that the bandwidth of PCIe is relatively slow and might be the bottleneck for data-intensive problems [36].

Based on the above analysis, three rules for heterogeneous computing is generated as follows:

- **Rule 1:** Allocate computational intensive task to GPU, and leave data-dependent conditional branches for CPU;
- **Rule 2:** Utilize the coalesced memory access pattern on device memory, and introduce the highly tuned and optimized libraries if it is possible;

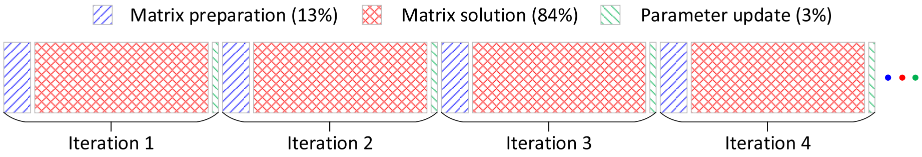


FIGURE 6. The execution time proportion of different operations of PDIPM.

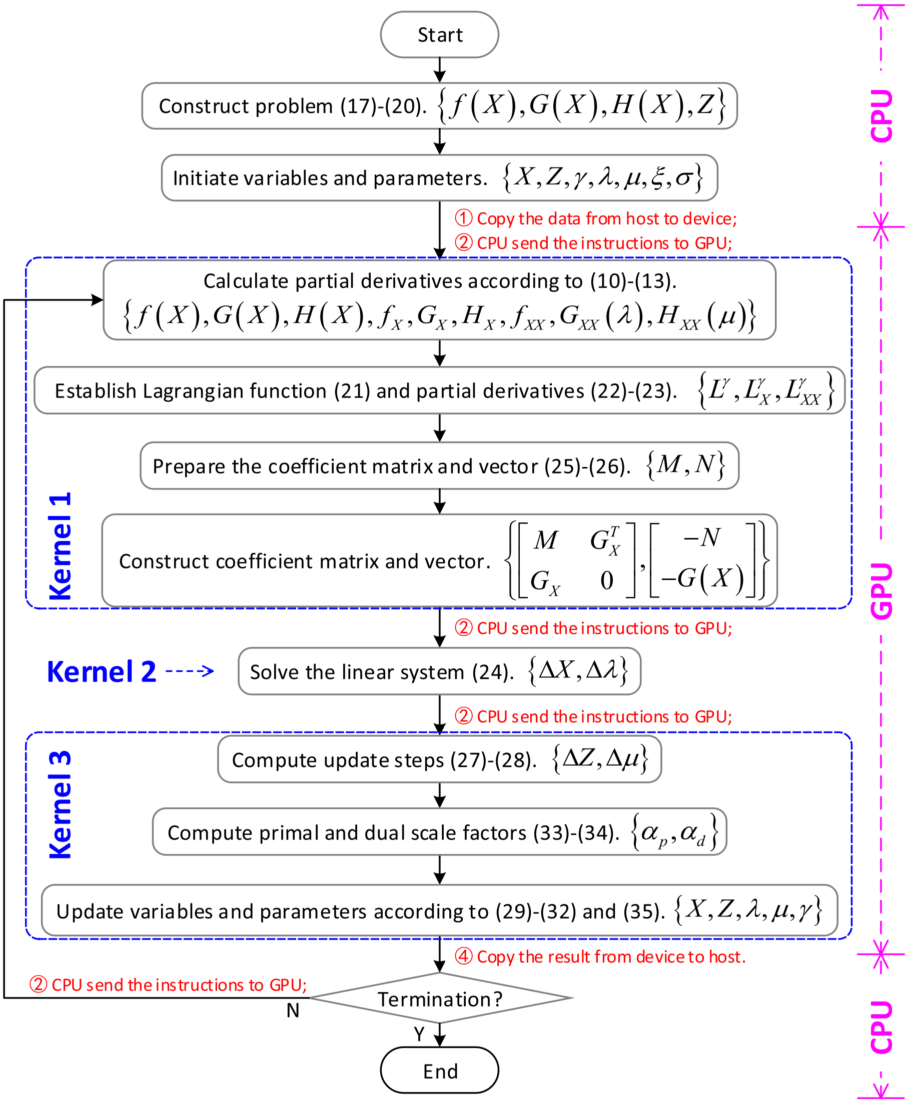

Start
Construct problem (17)-(20). $\{f(X),G(X),H(X),Z\}$
Initiate variables and parameters. $\{X,Z,\gamma,\lambda,\mu,\xi,\sigma\}$
① Copy the data from host to device;
② CPU send the instructions to GPU;
Calculate partial derivatives according to (10)-(13).
$\{f(X),G(X),H(X),f_X,G_X,H_X,f_{XX},G_{XX}(\lambda),H_{XX}(\mu)\}$
Establish Lagrangian function (21) and partial derivatives (22)-(23). $\{L^\gamma,L_X^\gamma,L_{XX}^\gamma\}$
Prepare the coefficient matrix and vector (25)-(26). $\{M,N\}$
Construct coefficient matrix and vector. $\left\{\begin{bmatrix}M & G_X^T\\ G_X & 0\end{bmatrix},\begin{bmatrix}-N\\ -G(X)\end{bmatrix}\right\}$
Kernel 1
② CPU send the instructions to GPU;
Kernel 2
Solve the linear system (24). $\{\Delta X,\Delta\lambda\}$
② CPU send the instructions to GPU;
Kernel 3
Compute update steps (27)-(28). $\{\Delta Z,\Delta\mu\}$
Compute primal and dual scale factors (33)-(34). $\{\alpha_p,\alpha_d\}$
Update variables and parameters according to (29)-(32) and (35). $\{X,Z,\lambda,\mu,\gamma\}$
④ Copy the result from device to host.
② CPU send the instructions to GPU;
Termination?
N
Y
End
CPU
GPU
CPU

FIGURE 7. Parallel implementation flowchart of concurrent PDIPM with heterogeneous architecture.

- **Rule 3:** Minimize the amount of data transferred between CPU and GPU.

C. IMPLEMENTATION FLOWCHART

According to Fig. 4, a set of RTOPFs corresponding to different scenarios should be solved in Stage 2. Since they are mutually independent, parallel implementation is introduced for concurrent solution in this subsection. Fig. 6 illustrates the proportion of execution time for different operations in a single PDIPM. It is obvious that the linear system (24) solution is the most intensive computational task, which consumes more than 80% of the whole execution time. Therefore, based on **Rule 1**, this process should be distributed to GPU. kernel 2 in Fig. 7 achieved this goal. Although the runtime of matrix preparation and parameter updating in Fig. 6 are less, and few data-dependent conditional branches are involved, they are executed on GPU with kernels 1 and 3 respectively

Algorithm 1 Horizontally Concatenating of Two Matrices $C = [A \mid B]$ in CSR Format (P, I, X, m, n, z)

```
1:  if $m_A \neq m_B$ then
2:      Return "dimension mismatch" message.
3:  else
4:      Let $m_C = m_A, n_C = n_A + n_B, z_C = z_A + z_B$;
5:      Set an indicator $ind = 1$;
6:      for $j = 1 \cdots m_A$ do
7:          for $p = P_A(j) + 1 \cdots P_A(j+1)$ do
8:              Let $I_C(ind) = I_A(p), X_C(ind) = X_A(p)$;
9:              Update $ind = ind + 1$;
10:         end for
11:         for $p = P_B(j) + 1 \cdots P_B(j+1)$ do
12:             Let $I_C(ind) = I_B(p) + n_A, X_C(ind) = X_B(p)$;
13:             Update $ind = ind + 1$;
14:         end for
15:         Let $P_C = P_A + P_B$ (this is a vector addition).
16:     end for
17: end if
```

Algorithm 2 Vertically Concatenating of Two Matrices $C = \left[\frac{A}{B}\right]$ in CSR Format (P, I, X, m, n, z)

```
1: if $n_A \neq n_B$ then
2:     Return "dimension mismatch" message.
3: else
4:     Let $n_C = n_A, m_C = m_A + m_B, z_C = z_A + z_B$;
5:     Let $I_C = [I_A, I_B], X_C = [X_A, X_B]$;
6:     Let $P_C = [P_A(1 \cdots m_A), P_A(m_A + 1) + P_B]$.
7: end if
```

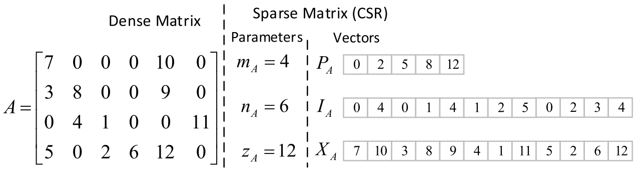

Dense Matrix
Sparse Matrix (CSR)
Parameters
Vectors
$A = \begin{bmatrix} 7 & 0 & 0 & 0 & 10 & 0 \\ 3 & 8 & 0 & 0 & 9 & 0 \\ 0 & 4 & 1 & 0 & 0 & 11 \\ 5 & 0 & 2 & 6 & 12 & 0 \end{bmatrix}$
$m_A = 4$
$n_A = 6$
$z_A = 12$
P_A 0 2 5 8 12
I_A 0 4 0 1 4 1 2 5 0 2 3 4
X_A 7 10 3 8 9 4 1 11 5 2 6 12

FIGURE 8. Demonstration of CSR format.

in Fig. 7 to minimize the data exchange between host and device (**Rule 3**), resulting in that the transferred data in each iteration is only the binary indicator for termination.

At the beginning, data preparation will be conducted on CPU. After data copying, kernel 1 will be launched by the instructions from CPU main thread. The execution of kernel 1 is simultaneous with blocks, where each block corresponds to one scenario. Within one block, a maximum number of 1024 threads can be called for parallel processing. In CUDA, there is an implicit barrier between different kernels. Therefore, kernel 2 will not be executed until the matrix preparation processes for all scenarios are finished. This barrier facilitates the utilization of batched solver for the solution of linear systems in kernel 2, which will be exemplified in the next subsection. Similarly, the barrier between kernels 2 and 3 guarantees all intermediate results from linear system solution have been updated for each scenario. The thread organization pattern in kernel 3 is the same with kernel 1. The termination judgment of (36) – (37) will be performed in kernel 3 and an indicator will be generated. Finally, CPU will copy that indicator from device to host memory when all blocks finished the calculation. The CPU will terminate the whole solution process if the termination criteria has been met; otherwise, execution instructions will be sent to kernel 1 by CPU to start a new iteration. It should be noted that all the intermediate data produced or updated by kernels are stored in device memory, thus the data exchange has been minimized and the barrier is required.

D. KERNELS DESIGN

In this paper, all vectors are dense, whereas all matrices are sparse and stored with Compressed Sparse Row (CSR) format as follows:

$$A = (P_A, I_A, X_A, m_A, n_A, z_A),$$

where m_A, n_A, and z_A are the numbers of rows, columns, and nonzero elements; P_A, I_A, and X_A are vectors with sizes of $m_A + 1$, z_A, and z_A, respectively. An illustrative example is given in Fig. 8.

1) KERNELS 1 AND 3

Data preparation and updating are the main tasks for kernels 1 and 3, where a lot of basic operations are involved, including matrix/vector element updating, matrix/vector addition, matrix-matrix/matrix-vector multiplication, matrix transposing, minimizing, and summation, etc. In order to achieve the best performance, a lot of investigation has been conducted to achieve the coalesced access pattern based on thread origination and shared memory utilization [39]–[41]. In this work, mature strategies reported in the literature are widely consulted and employed.

In addition to the basic operations, horizontally and vertically matrices concatenating are also involved in kernel 1, e.g., $\left[M \mid G_X^T\right]$ and $\left[\frac{M}{G_X}\right]$. **Algorithms** 1 and 2 illustrate the detailed steps of these two operations in the CSR format, whose parallel implementation is also conducted within each block based on the techniques discussed above. Line 6 in **Algorithm** 2 consists of the following operations: 1) $P_A(m_A + 1) + P_B$: add the $(m_A + 1)$th element of vector P_A into each entry of vector P_B; 2) $[P_A(1 \cdots m_A), P_A(m_A + 1) + P_B]$: combine the $(1 \cdots m_A)$ elements of vector P_A with the updated vector P_B into one whole vector.

2) KERNEL 2

As shown in Fig. 7, a set of linear systems with different coefficient matrices and vectors should be solved by kernel 2. For simplicity, they are denoted as $A_j x_j = b_j$ $(j = 1, 2, ..., N)$ in the following, where N is the number of linear systems. In order to follow the **Rule 2**, *cuSolver* library [42] is introduced for the solution of $A_j x_j = b_j$ on GPU. Table 1 summarizes different types of factorization methods provided by

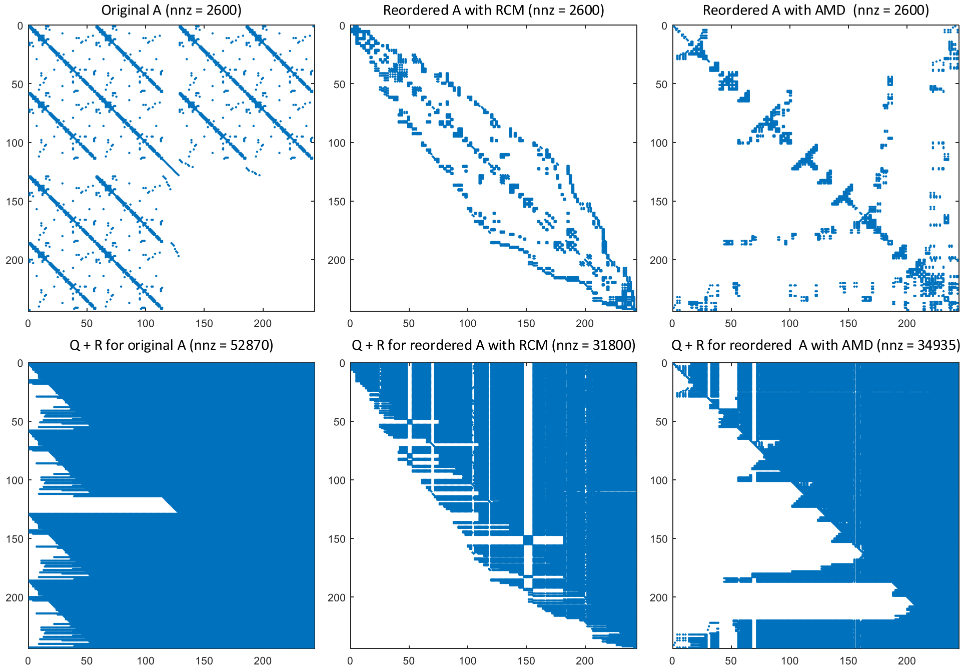


FIGURE 9. Performance illustration of RCM and AMD (X and Y axis correspond to the dimensions of each matrix; dot means that point is a nonzero entry).

TABLE 1. Accessible factorization methods of *cuSolver* in various modes.

Methods	CPU	GPU	
		regular mode	batched mode
Cholesky Factorization	√	√	×
LU Factorization	√	×	×
QR Factorization	√	√	√

cuSolver in various modes. In CPU and GPU regular mode, $A_jx_j = b_j$ will be solved one-by-one in *cuSolver* kernels. Although these kernels are highly optimized, the task switching between different linear systems is inevitable, resulting the whole solution efficiency is limited. On the other hand, the GPU batched mode can solve all the N linear systems $A_jx_j = b_j$ at the same time with only one call, thus full potential of GPU resources are utilized and the solution efficiency is highly improved. Based on the above analysis, the QR factorization is determined in this paper for the linear equations solution since it is the only method that supported by *cuSolver* in the GPU batched mode. The batched mode requires all the coefficient matrices have the same sparsity pattern, thus adjusting has been made on different matrices A_j in the same iteration as well as different iterations by adding zero values. By doing that, $P_{A_1} = P_{A_2} = \ldots = P_{A_N}$ and $I_{A_1} = I_{A_2} = \ldots = I_{A_N}$ can be achieved. The basic steps of batched QR solution is depicted as follows:

- **Prepare A_{batch}:** Summarize all the individual matrices A_j into one batched matrix A_{batch},

$$A_{batch} = \{P_{A_1}, I_{A_1}, X_{batch}, m_{A_1}, n_{A_1}, z_{A_1}\},$$

  where vector V_{batch} stores the nonzero elements of A_j one after another, i.e., $X_{batch} = \left[X_{A_1}, X_{A_2}, \ldots, X_{A_N}\right]$.
- **Generate permutation array q:** Generally, there will be extra nonzero entries named fill-ins appeared after the factorization, which demands extra memory space and more arithmetic operations. Fortunately, its number can be greatly reduced by matrix reordering, where a permutation array q is required. In this work, two algorithms, Reverse Chthill-Mckee (RCM) [43] and Approximate Minimum Degree (AMD) [44], are considered, whose intuitive performance is illustrated in Fig. 9, where the 'nnz' is the number of nonzero fill-ins.
- **Reorder A_{batch}:** The fill-ins reduction is performed by reordering A_{batch} into

$$B_{batch} = QA_{batch}Q^T,$$

  where Q is derived from q.
- **Symbolic analysis of B_{batch}:** This process is utilized to determine the sparsity pattern of lower and upper triangle matrices of QR factorization, which will be applied for the parallelism extraction and working space allocation.
- **Numerical factorization for B_{batch}:** This step is performed by all *CUDA cores* in the GPU with intensive parallelism. The generated solution x_{batch} should be reordered according to q before utilization in the following steps.

IV. CASE STUDIES

A. NETWORK AND INPUT DATA

The case studies are carried out on four benchmark systems modified from Matpower [45], including IEEE 14-bus, IEEE 57-bus, IEEE 118-bus, and IEEE 300-bus test cases, where 2, 4, 10, and 25 REGs are integrated to substitute thermal

TABLE 2. Execution time of RTOPF with different platforms (s).

Cases	14-bus	57-bus	118-bus	300-bus
Regular CPU	20.51	106.95	890.95	2,125.40
Parallel CPU	2.29	11.38	91.91	208.47
Regular GPU	1.51	5.88	43.54	85.05
Batched GPU	0.51	2.20	17.19	37.80

TABLE 3. Speedup of different methods over regular CPU implementation.

Cases	14-bus	57-bus	118-bus	300-bus
Regular CPU	1.00	1.00	1.00	1.00
Parallel CPU	8.95	9.40	9.69	10.20
Regular GPU	13.60	18.19	20.47	24.99
Batched GPU	40.22	48.61	51.83	56.23

generators. For each system, 1024 scenarios are generated at the beginning of each interval based on the meteorological data [46], REG parameter [19], and load profile [18], where time granularity adjusting, data normalization, and scenario reduction processes are employed.

All of these 1024 RTOPFs are tackled by a PC equipped with Intel Xeon E5-2620 CPU and Nvidia® GeForce GTX 1080 GPU. The programming environment is Visual Studio 2015 running on Windows 8.1 operating system. For simplicity, only one interval is conducted for comparison.

B. PERFORMANCE COMPARISON

In order to validate the performance of the framework developed in Section IV, four types of implementations are compared:

- Regular CPU: All scenarios are solved one-by-one, and the steps shown in Fig. 7 is sequentially executed for each scenario. In terms of linear equations solution, the sparse solver *Csparse* developed in [44] is utilized.
- Parallel CPU: All scenarios are solved in parallel based on the OpenMP Application Programming Interface (API) with each thread corresponding to one scenario. Within each scenario, the solution process is the same with regular CPU implementation, i.e., sequential and *Csparse*. In the case study, 12 threads are launched.
- Regular GPU: Parallel implementation on GPU according to the flowchart given in Fig. 7 except that the kernel 2 is performed by *cuSolver* QR factorization without batched mode enabled.
- Batched GPU: Parallel implementation on GPU with all the proposals shown in Section IV employed, including the batched QR factorization of *cuSolver*. The batch size is set at 1024 in case studies.

Although two types of fill-in reduction algorithms are considered and their performance on the 57-bus system shown in Fig. 9 is similar, the execution time of AMD is shorter in all test systems; therefore, the AMD is finally utilized for all the above four implementation schemes. Tables 2 and 3 summarize the main results for the solution time and speedup, where 1024 scenarios are considered for each test system.

1) RESULTS ON CPU PLATFORM

In [8], the execution for a 41-bus system with 49 scenarios is about 80s on CPU. Although the computation platforms are different, the results reported in Table 2 with regular CPU implementation are comparable, e.g., 106.95s is consumed for 57-bus system with 1024 scenarios. Nevertheless, it is far away from real-time application. Therefore, parallel computing with OpenMP is carried out on CPU with 12 threads enabled, resulting in the speedups from 8.95 to 10.20 for various test systems. It can be seen from Table 3 that the speedup is higher for larger system, which is partially due to the contradiction between overhead of thread switching and numerical calculation (larger system has heavier computation load). Fig. 10 demonstrates the achieved speedups for various systems with different numbers of threads launched. It is observable that the marginal profit gained by thread addition is diminishing, e.g., a speedup of 1.91 can be obtained by 2 threads for 14-bus system, whereas that number is only 8.95 for 12 threads. The reason is still related to the workloads of each scenario.

2) RESULTS ON GPU PLATFORM

Although GPU has a smaller frequency than CPU, the number of concurrent threads is much larger, thus the execution time is shorter than parallel CPU as shown in Table 2. In order to further improve the solution efficiency, batched mode is introduced, whose solution process is illustrated in Fig. 11 as well as the regular QR factorization. Both modes take 5 steps summarized in Section IV.D to solve a single or one bunch scenarios in iteration 1. Since the sparsity pattern of different coefficient matrices is tuned as the same on the batched QR, the results of reorder vector q and symbolic analysis are reusable, thus a lot of effort has been saved as highlighted with $\gg$ in Fig. 11. As reported in Table 3, the utilization of batched mode has doubled the speedup obtained by regular GPU implementation.

C. DISCUSSIONS

Apart from the above results on the solution time and speedup, the following discussions are given on two different topics.

1) BATCH SIZES

To intensively investigate the performance, different batch sizes are utilized for the solution of case 300-bus. The result is depicted in Fig. 12. The total execution time reduced quickly from 85.05s to 42.21s with the batch size increased from 1 to 256, validating the superiority of the batched mode. If the batch size keeps increasing, the runtime still decreases, but the rate is limited, indicating that the full potential of GPU is approaching. Since the fastest improvement occurs at the beginning rather than the later stage, one can use smaller batch size to increase the capability for larger systems with

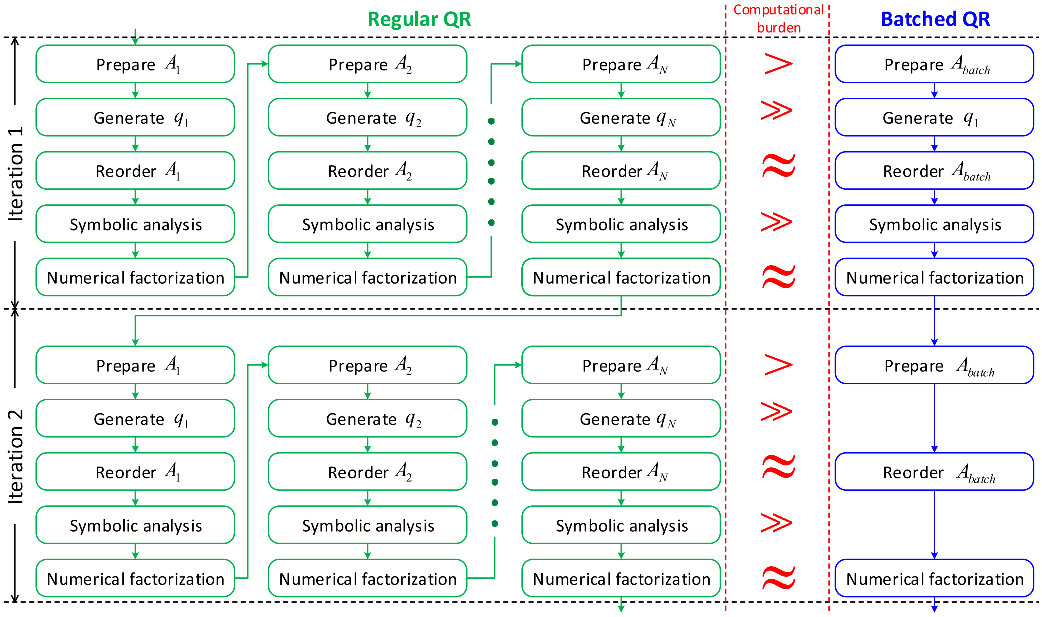


FIGURE 10. Achieved speedups by OpenMP on various test systems with different numbers of threads enabled.

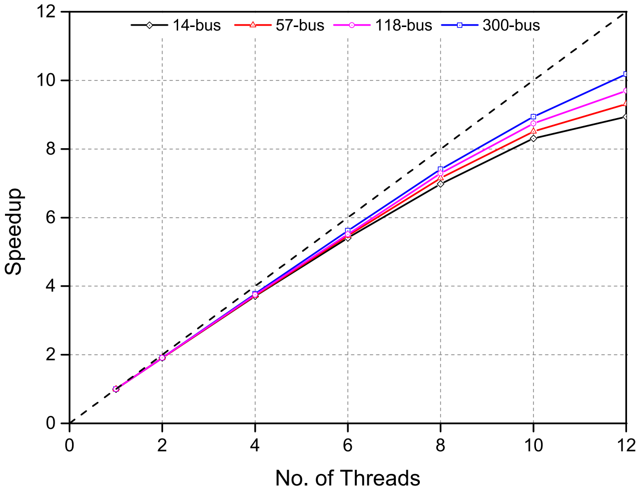


FIGURE 11. Solution process comparison between the regular and batched QR factorization.

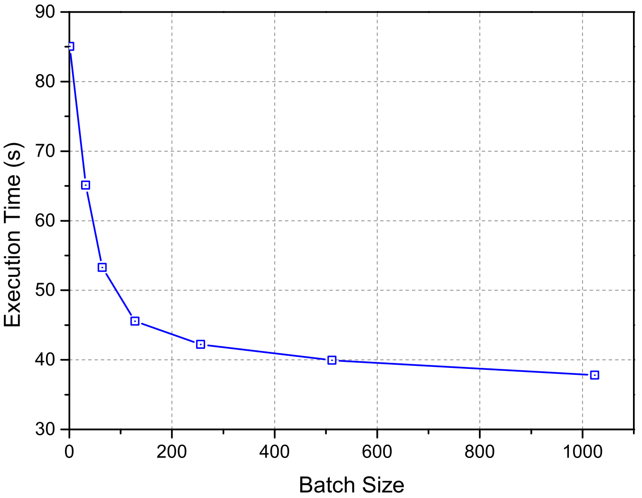


FIGURE 12. Execution time of case 300-bus with different batch sizes (1, 32, 64, 128, 256, 512, 1024).

TABLE 4. Effectiveness of the initiated solution based on different numbers of scenarios for the hot start linear system.

No. of scenarios	32	64	128	256	512	1024
No. of iterations for the final hot start linear system	8	5	3	2	1	1

little sacrifice of efficiency. For example, the solution time increased 11.67% by reducing the batch size from 2014 to 256 in Fig. 12.

2) HOT START

Table 4 summarizes the numbers of iterations for the final hot start linear system of case 118-bus on the basis of different numbers of scenarios. It is observable that the quality of the initiated solution is higher with more scenarios since the numbers of iterations before convergence is less. On the other hand, fewer iterations means that the initial solution is close to the final one, which can also be explained that the requirement for hot start strategy is more urgent for the circumstance with fewer scenarios. In order to gain better performance, one can increase the number of scenarios; however, there is a saturation point, after which the performance cannot be advanced. In the 118-bus case, that point is 512.

V. CONCLUSION

In order to minimize the forecasting error related with REGs and demand loads during the solution of RTOPF, a three-stage framework is employed in this paper. Scenarios are generated and filtered in Stage 1 to characterize the uncertainty. Stage 2 tries to minimize the prediction interval by GPU acceleration, where heterogeneous computing with batched linear solver is implemented. Hot-start strategy is introduced in Stage 3 to eliminate the prediction error. Comparison between CPU and heterogeneous CPU-GPU platforms are implemented on the IEEE 14-bus, 57-bus, 118-bus, and 300-bus systems, where both regular and batched solution schemes are included. The results validate the superiority of batched GPU over regular GPU, parallel CPU, and sequential CPU. Future work will investigate the saturation point related with scenario size and hot-start strategy, as well as other types of batched linear solver. Fuzzy-based methods for uncertainty addressing will also be considered.

APPENDIX
PRIMAL-DUAL INTERIOR POINT METHOD

In Section III, the nonlinear programming problem RTOPF is addressed with PDIPM [47]. In order to facilitate the description of GPU implementation, the solution process of PDIPM is reviewed in this section. The notations are summarized at the beginning, then the derivations are presented.

A. NOTATIONS

Given a real vector $X = [x_1, x_2, \ldots, x_n]^T$, the first (transpose of the gradient) and second partial (Hessian matrix) derivatives of a scalar function $f(X) : \mathbb{R}^n \to \mathbb{R}$ is given as:

$$f_X = \frac{\partial f}{\partial X} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right], \tag{10}$$

$$f_{XX} = \frac{\partial^2 f}{\partial X^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{11}$$

The vector function $F : \mathbb{R}^n \to \mathbb{R}^m$ is stated as $F(X) = [f_1(X), f_2(X), \ldots, f_m(X)]^T$, whose first derivatives (Jacobian matrix) is:

$$F_X = \frac{\partial F}{\partial X} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{12}$$

A matrix of partial derivatives of $F(X)$ based on the vector λ is:

$$F_{XX}(\lambda) = \frac{\partial(F_X{}^T \lambda)}{\partial X}. \tag{13}$$

Additionally, $[X]$ represents a diagonal matrix whose diagonal elements are valued by vector X, and e is the vector with all ones.

B. DERIVATIONS

For simplicity, the OPF problem (1)–(8) stated in Section II.B can be rewritten into a compact form as follows:

$$\min f(X), \tag{14}$$

$$s.t.\ G(X) = 0, \tag{15}$$

$$H(X) \le 0. \tag{16}$$

where X is the vector $[P_1^G, \ldots, P_n^G, Q_1^G, \ldots, Q_n^G, V_1, \ldots, V_m, \theta_1, \ldots, \theta_m]$; functions $f : \mathbb{R}^n \to \mathbb{R}$, $G : \mathbb{R}^n \to \mathbb{R}^p$, and $H : \mathbb{R}^n \to \mathbb{R}^q$ corresponds to (1), (2) – (3), and (4) – (7) and (9), respectively.

By introducing a barrier function, a perturbation parameter γ, and a positive slack vector Z, the problem is evolved into:

$$\min \left[f(X) - \gamma \sum_{i=1}^{q} \ln(Z_i)\right], \tag{17}$$

$$s.t.\ G(X) = 0, \tag{18}$$

$$H(X) + Z = 0, \tag{19}$$

$$Z > 0. \tag{20}$$

The Lagrangian function of this equality constrained problem (17)–(20) is:

$$L^\gamma = f(X) + \lambda^T G(X) + \mu^T (H(X) + Z) - \gamma \sum_{i=1}^{q} \ln(Z_i). \tag{21}$$

The first and second particle derivatives of (21) over X is given as:

$$L_X^\gamma = f_X + G_X \lambda + H_X \mu, \tag{22}$$

$$L_{XX}^\gamma = f_{XX} + G_{XX}(\lambda) + H_{XX}(\mu). \tag{23}$$

Based on the Karush-Kuhn-Tucker conditions and Newton's method, the following iterative updating procedure can be performed with initiated X, Z, λ, μ, and γ:

- **Step 1:** Compute ΔX and $\Delta\lambda$ according to:

$$\begin{bmatrix} M & G_X{}^T \\ G_X & 0 \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -N \\ -G(X) \end{bmatrix}, \tag{24}$$

where

$$M = L_{XX}^\gamma + H_X{}^T [Z]^{-1} [\mu] H_X, \tag{25}$$

$$N = L_X^{\gamma\, T} + H_X{}^T [Z]^{-1} (\gamma e + [\mu] H(X)). \tag{26}$$

- **Step 2:** Compute ΔZ and $\Delta\mu$ from:

$$\Delta Z = -H(X) - Z - H_X \Delta X, \tag{27}$$

$$\Delta \mu = -\mu + [Z]^{-1} (\gamma e - [\mu] \Delta Z). \tag{28}$$

- **Step 3:** Update variables X, Z, λ, and μ according to:

$$X = X + \alpha_p \Delta X, \tag{29}$$

$$Z = Z + \alpha_p \Delta Z, \tag{30}$$

$$\lambda = \lambda + \alpha_d \Delta \lambda, \tag{31}$$

$$\mu = \mu + \alpha_d \Delta \mu. \tag{32}$$

where the primal and dual scale factors α_p and α_d is derived by the following equations with constant parameter $\xi = 0.99995$:

$$\alpha_p = \min\left(\xi \min_{\Delta Z_i<0}\left(-\frac{Z_i}{\Delta Z_i}\right), 1\right), \tag{33}$$

$$\alpha_d = \min\left(\xi \min_{\Delta \mu_i<0}\left(-\frac{\mu_i}{\Delta \mu_i}\right), 1\right). \tag{34}$$

- **Step 4:** Update perturbation parameter γ by the following mechanism, where σ is set to 0.1:

$$\gamma = \sigma \frac{Z^T\mu}{q}. \tag{35}$$

- **Step 5:** Terminate the solution process if convergent criteria (36) – (37) are met, where f and f_0 are the objective function value of the current and previous iterations respectively; ϵ is valued as 1.0×10^{-6}. Otherwise, go to **Step 1**.

$$\frac{\max\{\|G(X)\|_\infty, \|H(X)\|_\infty\}}{1+\max\{\|X\|_\infty, \|Z\|_\infty\}} < \epsilon, \quad \frac{|f-f_0|}{1+|f_0|} < \epsilon, \tag{36}$$

$$\frac{\|L_X^\gamma\|_\infty}{1+\max\{\|\lambda\|_\infty, \|\mu\|_\infty\}} < \epsilon, \quad \frac{Z'\mu}{1+\|X\|_\infty} < \epsilon. \tag{37}$$

REFERENCES

[1] J. Carpentier, "Contribution a letude du dispatching economique," *Bull. Soc. Francaise Elect.*, vol. 3, no. 1, pp. 431–447, Aug. 1962.

[2] R. Bacher and H. P. Van Meeteren, "Real-time optimal power flow in automatic generation control," *IEEE Trans. Power Syst.*, vol. 3, no. 4, pp. 1518–1529, Nov. 1988.

[3] S. Xia, X. Luo, K. W. Chan, M. Zhou, and G. Li, "Probabilistic transient stability constrained optimal power flow for power systems with multiple correlated uncertain wind generations," *IEEE Trans. Sustain. Energy*, vol. 7, no. 3, pp. 1133–1144, Jul. 2016.

[4] D. Ke, C. Y. Chung, and Y. Sun, "A novel probabilistic optimal power flow model with uncertain wind power generation described by customized Gaussian mixture model," *IEEE Trans. Sustain. Energy*, vol. 7, no. 1, pp. 200–212, Jan. 2016.

[5] Z.-S. Zhang, Y.-Z. Sun, D. W. Gao, J. Lin, and L. Cheng, "A versatile probability distribution model for wind power forecast errors and its application in economic dispatch," *IEEE Trans. Power Syst.*, vol. 28, no. 3, pp. 3114–3125, Aug. 2013.

[6] S. S. Reddy and P. R. Bijwe, "Day-ahead and real time optimal power flow considering renewable energy resources," *Int. J. Electr. Power Energy Syst.*, vol. 82, pp. 400–408, Nov. 2016.

[7] E. Mohagheghi, A. Gabash, and P. Li, "Real-time optimal power flow under wind energy penetration—Part I: Approach," in *Proc. IEEE Int. Conf. Environ. Elect. Eng.*, Florence, Italy, Jun. 2016, pp. 1–6.

[8] E. Mohagheghi, A. Gabash, and P. Li, "Real-time optimal power flow under wind energy penetration—Part II: Implementation," in *Proc. IEEE Int. Conf. Environ. Elect. Eng.*, Florence, Italy, Jun. 2016, pp. 1–6.

[9] R. J. Bessa, V. Miranda, A. Botterud, J. Wang, and E. M. Constantinescu, "Time adaptive conditional kernel density estimation for wind power forecasting," *IEEE Trans. Sustain. Energy*, vol. 3, no. 4, pp. 660–669, Oct. 2012.

[10] W. He and S. S. Ge, "Cooperative control of a nonuniform gantry crane with constrained tension," *Automatica*, vol. 66, no. 4, pp. 146–154, Apr. 2016.

[11] A. Webberley and D. W. Gao, "Study of artificial neural network based short term load forecasting," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, Vancouver, BC, Canada, Jul. 2013, pp. 1–4.

[12] W. He, Y. Chen, and Z. Yin, "Adaptive neural network control of an uncertain robot with full-state constraints," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 620–629, Mar. 2016.

[13] W. He, Z. Yan, C. Sun, and Y. Chen, "Adaptive neural network control of a flapping wing micro aerial vehicle with disturbance observer," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3452–3465, Oct. 2017.

[14] X. Xie, D. Yue, H. Zhang, and C. Peng, "Control synthesis of discrete-time T–S fuzzy systems: Reducing the conservatism whilst alleviating the computational burden," *IEEE Trans. Cybern.*, vol. 47, no. 9, pp. 2480–2491, Sep. 2017.

[15] J. R. Andrade and R. J. Bessa, "Improving renewable energy forecasting with a grid of numerical weather predictions," *IEEE Trans. Sustain. Energy*, vol. 8, no. 4, pp. 1571–1580, Oct. 2017.

[16] L. Gan and S. H. Low, "An online gradient algorithm for optimal power flow on radial networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 625–638, Mar. 2016.

[17] S. S. Reddy and J. A. Momoh, "Realistic and transparent optimum scheduling strategy for hybrid power system," *IEEE Trans. Smart Grid*, vol. 6, no. 6, pp. 3114–3125, Nov. 2015.

[18] Y. Tang, K. Dvijotham, and S. Low, "Real-time optimal power flow," *IEEE Trans. Smart Grid*, vol. 8, no. 6, pp. 2963–2973, Nov. 2017.

[19] E. Mohagheghi, A. Gabash, and P. Li, "A framework for real-time optimal power flow under wind energy penetration," *Energies*, vol. 10, no. 4, p. 535, Apr. 2017.

[20] V. Jalili-Marandi and V. Dinavahi, "SIMD-based large-scale transient stability simulation on the graphics processing unit," *IEEE Trans. Power Syst.*, vol. 25, no. 3, pp. 1589–1599, Aug. 2010.

[21] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 7, pp. 1255–1266, Jul. 2012.

[22] Z. Zhou and V. Dinavahi, "Parallel massive-thread electromagnetic transient simulation on GPU," *IEEE Trans. Power Del.*, vol. 29, no. 3, pp. 1045–1053, Jun. 2014.

[23] Z. Zhou and V. Dinavahi, "Fine-grained network decomposition for massively parallel electromagnetic transient simulation of large power systems," *IEEE Power Energy Technol. Syst. J.*, vol. 4, no. 3, pp. 51–64, Sep. 2017.

[24] P. Liu and V. Dinavahi, "Finite-difference relaxation for parallel computation of ionized field of HVDC lines," *IEEE Trans. Power Del.*, vol. 33, no. 1, pp. 119–129, Feb. 2018.

[25] H. Karimipour and V. Dinavahi, "Extended Kalman filter-based parallel dynamic state estimation," *IEEE Trans. Smart Grid*, vol. 6, no. 3, pp. 1539–1549, May 2015.

[26] H. Karimipour and V. Dinavahi, "Parallel relaxation-based joint dynamic state estimation of large-scale power systems," *IET Gen., Transm. Distrib.*, vol. 10, no. 2, pp. 452–459, Feb. 2016.

[27] L. Rakai and W. Rosehart, "GPU-accelerated solutions to optimal power flow problems," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, Waikoloa, HI, USA, Jan. 2014, pp. 2511–2516.

[28] G. Geng, Q. Jiang, and Y. Sun, "Parallel transient stability-constrained optimal power flow using GPU as coprocessor," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1436–1445, May 2017.

[29] J. Zhu, *Optimization of Power System Operation*, 2nd ed. Hoboken, NJ, USA: Wiley, 2015.

[30] J. Lavaei and S. H. Low, "Zero duality gap in optimal power flow problem," *IEEE Trans. Power Syst.*, vol. 27, no. 1, pp. 92–107, Feb. 2012.

[31] NREL Solar Radiation Research Laboratory. (Jan. 1, 2015/Nov. 30, 2017). *Daily Plots and Raw Data Files*. [Online]. Available: http://midcdmz.nrel.gov/srrl_bms/

[32] W. He and S. S. Ge, "Vibration control of a nonuniform wind turbine tower via disturbance observer," *IEEE/ASME Trans. Mechatron.*, vol. 20, no. 1, pp. 237–244, Feb. 2015.

[33] Y. Wang, Y. Liu, and D. S. Kirschen, "Scenario reduction with submodular optimization," *IEEE Trans. Power Syst.*, vol. 32, no. 3, pp. 2479–2480, May 2017.

[34] E. Mohagheghi, A. Gabash, and P. Li, "A study of uncertain wind power in active-reactive optimal power flow," in *Proc. Power Energy Stud. Summit*, Dortmund, Germany, Jan. 2016, pp. 1–6.

[35] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[36] *CUDA C Programming Guide 8.0*, NVIDIA, Santa Clara, CA, USA, 2017.

[37] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, and A. Shringarpure, "On the limits of GPU acceleration," in *Proc. USENIX Conf. Hot Topics Parallelism*, Berkeley, CA, USA, 2010, pp. 1–6.

[38] G. Zhou *et al.*, "GPU-accelerated batch-ACPF solution for N-1 static security analysis," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1406–1416, May 2017.

[39] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Corp., Santa Clara, CA, USA, Tech. Rep. NVR-2008-004, 2008.

[40] M. Aleem, R. Prodan, and T. Fahringer, "On the extension and evaluation of the JavaSymphony for heterogeneous parallel computers," in *Proc. Int. Conf. Parallel Process.*, Pittsburgh, PA, USA, Nov. 2012, pp. 1–11.

[41] D. Merrill and M. Garland, "Merge-based sparse matrix-vector multiplication (SpMV) using the CSR storage format," in *Proc. ACM SIGPLAN Symp. Principles Pract. Parallel Program.*, Barcelona, Spain, Mar. 2016, pp. 1–2.

[42] *CUSOLVER Library 8.0*, NVIDIA, Santa Clara, CA, USA, 2017.

[43] J. R. Gilbert, C. Moler, and R. Schreiber, "Sparse matrices in MATLAB: Design and implementation," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 333–356, Jan. 1992.

[44] T. A. Davis, *Direct Methods for Sparse Linear Systems*. Philadelphia, PA, USA: SIAM, 2006.

[45] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[46] NREL. *Measurement and Instrumentation Data Center (MIDC)*. Accessed: Aug. 9, 2017. [Online]. Available: http://www.nrel.gov/midc/

[47] H. Wang, C. E. Murillo-Sanchez, R. D. Zimmerman, and R. J. Thomas, "On computational issues of market-based optimal power flow," *IEEE Trans. Power Syst.*, vol. 22, no. 3, pp. 1185–1193, Aug. 2007.

SHENGJUN HUANG (S'14) received the B.Sc. and M.Sc. degrees in management science and engineering from the National University of Defense Technology, Changsha, China, in 2011 and 2013. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His research interests include mixed-integer linear programming, decomposition algorithms, large-scale power systems, and parallel computing.

VENKATA DINAVAHI (SM'08) received the Ph.D. degree in electrical and computer engineering from the University of Toronto, Canada, in 2000. He is currently a Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His research interests include real-time simulation of power systems and power electronic systems, large-scale system simulation, and parallel and distributed computing.