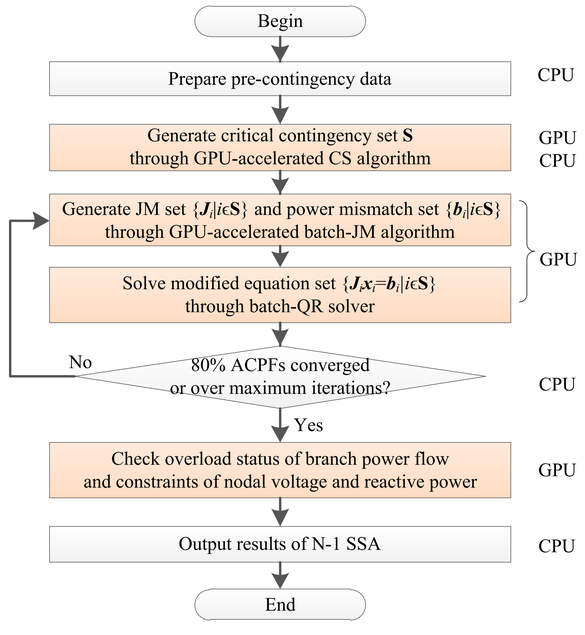
# GPU-Accelerated Batch-ACPF Solution for N-1 Static Security Analysis

Gan Zhou, Yanjun Feng, Xu Zhang
Southeast University

## Introduction

- The N-1 static security analysis (SSA) is used to ensure steady-state security of power system under the loss of any single element. For a system with *N* elements, **a strict N-1 SSA requires solving *N* alternating-current power flows (ACPF)**
- Several **sparse linear systems (SLS)** need to be calculated by Newton's method in each ACPF
- Even with the contingency screening (CS), the N-1 SSA is still a computational intensive task. For example, the East China power grid has **8,503 buses** and 12,475 expected contingencies. In a test platform with an Intel Xeon 2G Hz CPU, it takes about **2.4 minutes** to solve a SSA with only a CCS of **475 contingencies**, which cannot meet the speed requirement of on-line analysis.
- This poster presents a GPU-Accelerated **Batch-ACPF Solution** for N-1 Static Security Analysis, which is different from GPU-accelerated **sequential-ACPF solution** who uses GPU to accelerating ACPF one by one

## Batch-ACPF Solution for SSA Overview




## Analysis of GPU-accelerated sequential-ACPF solution

When GPU technique is employed, one natural solution is utilizing it to accelerate ACPFs one by one, which is termed sequential-ACPF solution in this poster. More specifically, each contingency in the CCS is chosen one after another to carry out GPU-based ACPF analysis. The potential bottlenecks with this solution include:

- For a mainstream HPC-purposed GPU such as NVIDIA Tesla K20 or K40, a one-million-order SLS is considered as a small-to-medium-scale computation problem. **In contrast, the power system JM is typically less than 0.1 million order and so solving this ACPF is only a small-scale calculation problem.** Moreover, the numerical factorization process for JM is excessively sequential, so it does not create enough parallelism to saturate the numerous cores in HPC-purposed GPU.
- Even if an ACPF solving problem has enough parallelism to keep all GPU cores utilized, **random memory access resulted from sparse numerical factorization** will lead to massive uncoalesced memory access and therefore the high bandwidth of GPU cannot be fully utilized.

## GPU-Accelerated Batch-QR Solver

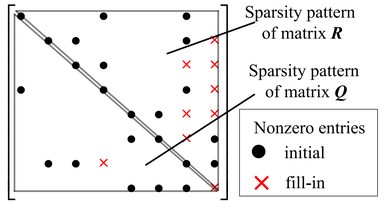
### A Choosing QR factorizing method

QR factorization is absolutely numerical stable even **without pivoting** and so it is more suitable for solving GPU-based batch-SLS.

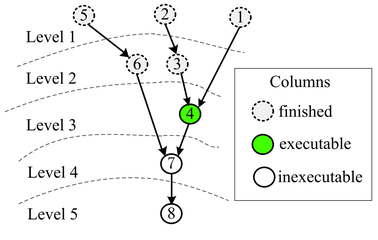
### B QR solver for a single SLS

**Parallel sparse left-looking QR factorization Algorithm**

**for** $i$ = 1 : $n$ **do //** $n$ is the number of levels in dependence graph ***T***.
**While** there are still unfinished columns **do in parallel**
  Factorize column according to ***T***;
  Generate Householder vector;
Modify right side vector;
**end while**
**end for**

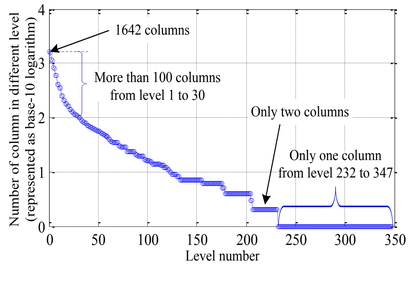



Sparsity pattern with fill-ins



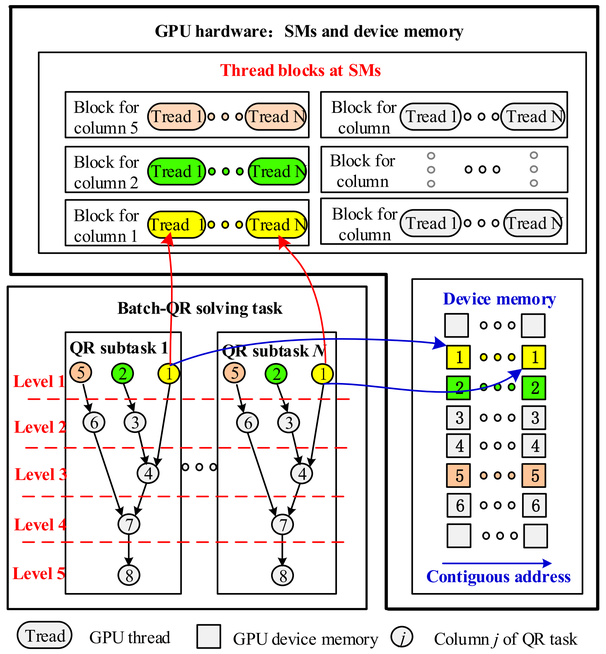

Dependence graph between different columns.

### Level information of dependence graph on an 8503-bus case




The parallelism of Algorithm performs worse and worse with the execution of QR factorizing and the random memory access caused by spares QR factorization is very difficult to optimize

### C Batch-QR solver for massive number of SLSs




The poster presents a novel batch-QR solving method which can solve a set of linear systems $\boldsymbol{A}_i\boldsymbol{x}_i=\boldsymbol{b}_i$ ($i$=1, 2, …, $N$) simultaneously where each nonsingular $\boldsymbol{A}_i$ must have the same sparsity pattern, $\boldsymbol{x}_i$ and $\boldsymbol{b}_i$ are dense vectors.

And as any contingency influences at most four elements of pre-contingency bus admittance matrix $\boldsymbol{Y}_0$, the sparsity pattern of any post-contingency admittance matrix $\boldsymbol{Y}_i$ can be designed as same as $\boldsymbol{Y}_0$. Similarly, the sparsity pattern of any post-contingency JM $J_i$ can be designed as same as pre-contingency JM $J_0$.

**In this way, Batch-ACPF will be launched successfully, and the Batch-QR Solver overcome the shortcomings of Single QR solving.**

## Performance

Performance Analysis of Batch-ACPF Solution on 8503-bus system

| Contingency set size | CPU | GPU-accelerated | | | |
|---|---|---|---|---|---|
| | Sequential ACPF | Sequential ACPF | | Batch ACPF | |
| | Time (s) | Time (s) | Speedup | Time (s) | Speedup |
| 141 | 47.2 | 37.8 | 1.25× | 1.99 | 23.7× |
| 266 | 85.8 | 67.9 | 1.26× | 2.13 | 40.2× |
| 457 | 144.8 | 114.0 | 1.27× | 2.51 | 57.6× |
| 623 | 196.1 | 153.9 | 1.27× | 3.02 | 64.9× |

When the number of critical contingencies $S_{cs}$=457, the average computation time per ACPF is reduced from 309ms to only 4.43 ms and up to 69.7 times speedup is achieved in comparison to MATPOWER-based CPU counterpart.

The proposed GPU-accelerated batch-ACPF solution is very promising and can be widely used **in other power system applications** which deal with massive number of homogeneous subtasks, such as **Monte-Carlo simulation and probabilistic power flow**.