Power System Probabilistic and Security Analysis Using Commodity High Performance Computing Systems

Tao Cui (Presenter)
Franz Franchetti
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
tao.cui@ieee.org, franzf@ece.cmu.edu

This work was supported by NSF through awards 0931978 and 1116802.
Smart Grid Challenges

**The Source: Probabilistic**

- New players: stochastic, varying renewable

- Challenges
  - Non-dispatchable, stochastic in nature, large variances
  - Great impact on the grids: present and future trend

**The Grid: Security**

- Aged, varying conditions & close to limit operations

- Challenges
  - High contingency possibility w/ severe consequences
  - Varying grid conditions require merging offline to online

**Requirements from regulators (NERC)**

- Probabilistic analysis approaches from distribution feeder to bulk grid\(^N_1\)
- Extensive contingency analysis for online / real time\(^N_2\)

**Online computation tool for the smart grid challenges**

---

\(^N_1\) NERC Special Report: Accommodating High Levels of Variable Generation, 2009
\(^N_2\) NERC Transmission System Standards – Normal and Emergency Conditions
Commodity CPUs: No Free Speedup

- Top 500 HPC in every substation?

- Peak performance = fully utilize hardware capability
- Much more complicated architecture ~ lose performance easily

Performance in “flop/s”: floating point operations per second, the higher the better
Commodity CPUs: Opportunity & Challenge

- Memory hierarchy: #cache memory system
- Multilevel parallelism:
  - *Multi-thread on multi-core CPU
  - ^Superscalar instruction scheduler: parallel scheduler inside each core
  - $SIMD (Single Instruction Multiple Data): Intel AVX/SSE instruction set
Challenges for Smart Grids Applications

- **Computing**
  - Exponential growth (Moore’s law)
  - Peak perf: fully using all computing units
  - Lose performance easily

- **Numerical application development**
  - Example: MMM on Core 2 Duo
  - Gap between SW/HW for power system
    - Performance tuning for HW
    - Best math library? or best algorithm?

**Goal: a win-win result**

*Fully* utilize the modern commodity computing systems, make new important and difficult smart grid probabilistic and security analysis problems solvable... in real time.
Content

- Distribution feeder probabilistic load flow (DPLF)
- Transmission grid probabilistic load flow (TPLF)
- AC contingency calculation (ACCC)

Goal: a win-win result
Fully utilize the modern commodity computing systems, make new important and difficult smart grid probabilistic and security analysis problems solvable... in real time.
Outline: HPC in Substation

- Distribution feeder probabilistic load flow (DPLF)
- Transmission grid probabilistic load flow (TPLF)
- AC contingency calculation (ACCC)
Distribution PLF: Background

- **Fundamental tool for probabilistic analysis**
  - Wind in distribution systems [Jorgensen:1998] [Caramia:2007]
  - Solar in distribution systems [Ruiz:2012]
  - EV in distribution systems [Li:2012]
  - Monte Carlo simulation as gold standard

- **Unique challenges in distribution systems**
  - Unbalanced, multi-phase, complicated device models
  - Nonlinearity: regulator, control devices
  - Monte Carlo simulation: robust, generally-applicable, UNFEASIBLE online?

- **Our approaches** [Cui_PES:2012]
  - Hardware / software performance engineering perspective
  - Pushing Monte Carlo simulation practical for online operations

---

[Tao Cui](mailto:taocui@cmu.edu) and Franz Franchetti.
Our Approach

- Random number generator
  - Basic uniform Pseudo RNG + transformation for different PDFs
  - Parallel strategy for multi-thread implementation

- Optimized parallel distribution load flow solver
  - Code optimizations
  - Highly parallel implementation for Monte Carlo applications

- Density estimation & visualization
  - Kernel density estimation
Load Flow Algorithm

- **Distribution system:**
  - Radial, high R/X ratio, varying Z, unbalanced
  - **NOT** suitable for transmission load flow
  - IV Forward / Backward Sweep (FBS) [Zimmer:1995]
    - Implicit Z-matrix, detail model, fewest flops

- **Generalized component model** [Kersting:2006]
  - One terminal node model:
    - Constant PQ:
      \[
      [S_{abc}] = [V_{abc}] \cdot [I_{abc}]^*
      \]
  - Two terminal link model:
    - Backward:
      \[
      [I_{abc}]_n = [c] \cdot [V_{abc}]_m + [d] \cdot [I_{abc}]_m
      \]
    - Forward:
      \[
      [V_{abc}]_m = [A] \cdot [V_{abc}]_n - [B] \cdot [I_{abc}]_m
      \]

---


Code Optimization

- **Data structure optimization**
  - Transform **Baseline C++** to array access, exploit spatial/temporal locality
  - Transform matrix-vector multiplication for Kersting’s algorithm
  - Reduce unnecessary operations
  - Unrolling innermost loops
  - Similar to [Belgin:2009]: pattern SpMV

**Algorithm-level optimization: pattern & unrolling**

- Pattern based matrix-vector multiplication
  \[ [I_{abc}]_n = [c] \cdot [V_{abc}]_m + [d] \cdot [I_{abc}]_m \]
  \[ [V_{abc}]_m = [A] \cdot [V_{abc}]_n - [B] \cdot [I_{abc}]_m \]
- Reduce unnecessary operations
- Unrolling innermost loops

```
switch (mat_type){
  case real_diag_equal_mat:
    output[0] = *constant * input[0];
    ...
    output[5] = *constant * input[5];
    break;
  case imag_diag_equal_mat:
    output[0] = -*constant * input[3];
    output[1] = -*constant * input[4];
    output[2] = -*constant * input[5];
    output[3] = *constant * input[0];
    output[4] = *constant * input[1];
    output[5] = *constant * input[2];
    break;
  ...
}
```

Parallel MCS with Real Time Considerations

- **Data level (SIMD)**

- **Task/core level**

- Realtime MCS scheduler: task decomposition, double buffer, auto-balancing)

- Fully utilize computation power of multi-core CPUs
## Details: Performance Gains

### Performance Impact of Optimization & Parallelization on Core i7

![Graph showing performance gains](image)

- **flop/s: >60 % peak**

### Problem Size (IEEE Test Feeders)

<table>
<thead>
<tr>
<th></th>
<th>Approx.</th>
<th>Approx. Time /</th>
<th>Approx. Time /</th>
<th>Baseline. C++ ICC –o3 (~300x faster then pure Matlab scripts)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>flops</td>
<td>Core2 Extreme</td>
<td>Core i7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEEE37: one iteration</td>
<td>12 K</td>
<td>~ 0.3 us</td>
<td>~ 0.3 us</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEEE37: one load flow (5 Iter)</td>
<td>60 K</td>
<td>~ 1.5 us</td>
<td>~ 1.5 us</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEEE37: 1 million load flow</td>
<td>60 G</td>
<td>~ &lt; 2 s</td>
<td>~ &lt; 1 s</td>
<td>~ 60 s (&gt;5 hrs Matlab)</td>
<td>SCADA Interval: 4 seconds</td>
</tr>
<tr>
<td>IEEE123: 1 million load flow</td>
<td>200 G</td>
<td>~ &lt; 10 s</td>
<td>~ &lt; 3.5 s</td>
<td>~ 200 s (&gt;15 hrs Matlab)</td>
<td></td>
</tr>
</tbody>
</table>

- **Core i7 2670QM, quad-core, 2.2GHz, AVX, Machine Peak: 140 Gflop/s**
- **Millions of load flow cases solved within SCADA interval**
## Accuracy

### Convergence of Monte Carlo

- **Crude MCS: MCG59+ICDF, 50 trials with “time(NULL)” seeds**

  - 100 run
  - 1k run
  - 10k run
  - 100k run
  - 1M run
  - 10M run

- **Converged MCS solution to PLF within SCADA Interval**
  - Accurate, generally-applicable, real time

- **In:** Active Power $P \sim u=0, \text{std}=100\text{kw}$ on Phase A of Node 738,711,741

- **Out:** Voltage on Node 738

\[ 0 \leq v_{\text{p.u.}} \leq 1 \]
Applications

- Distribution System Probabilistic Monitoring System (DSPMS)
  - System structure:
    - **Input:** smart meters in campus building (MODBUS/TCP)
    - **Solver:** MCS solver on multi-core desktop server (code optimization)
    - **Output:** dynamic web interface via ECE web server (TCP/IP, JavaScript)
User Interfaces

- **Left:** application 1 -- online forecasting (e.g. Gaussian error)
- **Right:** application 2 -- online impact assessment

- Live update, every 4 seconds (SCADA interval)
- Novel SCADA extension with full real time probabilistic analysis
Outline: HPC in Substation

- Distribution feeder probabilistic load flow (DPLF)
- Transmission grid probabilistic load flow (TPLF)
- AC contingency calculation (ACCC)
Related work on improving PLF solutions

- Simplified power system model:
  - linearized, multi-linearized \cite{Allan:1981,Silva:1990}

- Simplified probabilistic model:
  - Interval \cite{Wang:1992}, point \cite{Su:2005}, cumulants \cite{Zhang:2004}

- Improving Monte Carlo
  - Variance reduction \cite{Zhang:2009}, deterministic sample \cite{Liao:2007}

- Monte Carlo simulation as accuracy references

Opportunities from computing hardware perspective

- Analytical method: not robust, not generally-applicable...
- Fully take advantages of the computing system
- Can we push the gold standard (MCS) into real time?
Fast Decoupled Power Flow

- Fast, fewest floating point operations

\[
\begin{aligned}
\Delta V^{(k)} &= -B''^{-1} \Delta Q(\theta^{(k)}, V^{(k)}) / V^{(k)} \\
V^{(k+1)} &= V^{(k)} + \Delta V^{(k)}
\end{aligned}
\]

\[
\begin{aligned}
\Delta \theta^{(k)} &= -B'^{-1} \Delta P(\theta^{(k)}, V^{(k+1)}) / V^{(k+1)} \\
\theta^{(k+1)} &= \theta^{(k)} + \Delta \theta^{(k)}
\end{aligned}
\]

\[
\begin{aligned}
\Delta P_i &= P_i^{INJ} - V_i \sum_{j=1}^{n} V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij}) \\
\Delta Q_i &= Q_i^{INJ} - V_i \sum_{j=1}^{n} V_j (G_{ij} \sin \theta_{ij} - B_{ij} \cos \theta_{ij})
\end{aligned}
\]

[Stott:1974] Stott, B.
Review of load-flow calculation methods
Proceedings of the IEEE, 1974, 62, 916 - 929

"Fast decoupled load flow: hypothesis, derivations, and testing,"
Algorithm Optimization: Sparsity

- Linear solver: exploit sparsity -- Baseline
  - Sparse LU factors by using AMD\[^{[Davis:2004]}\]
  - Speedup Matpower 4.1’s FDPF module

Original L’
Original U’
Sparse L’ by AMD
Sparse U’ by AMD
LU Factors of B’ of IEEE 118 system

Speed Comparison

\[^{[Davis:2004]}\] P. Amestoy, T. Davis, and I. Duff, 
“Algorithm 837: AMD, an approximate minimum degree ordering algorithm,”
ACM Transactions on Mathematical Software (TOMS), vol. 30, no. 3, pp. 381–388, 2004
Data Structure/Math Func Optimization

- **Linear solver**
  - New mixed compress column storage (CCS) format

<table>
<thead>
<tr>
<th>col ptr</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>row idx</td>
<td>r1</td>
<td>r2</td>
<td>r3</td>
</tr>
<tr>
<td>value:</td>
<td>v1</td>
<td>v2</td>
<td>v3</td>
</tr>
</tbody>
</table>

  - Mixed CCS:

<table>
<thead>
<tr>
<th>col ptr</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mixed:</td>
<td>v1</td>
<td>r1</td>
<td>v2</td>
</tr>
</tbody>
</table>

- **Mismatch**

  - Original θ array: \( \theta_1 \) \( \theta_2 \) ... \( \theta_N \)
  - Mixed θ array: \( \theta_1 \) sin\( \theta_1 \) cos\( \theta_1 \) \( \theta_2 \) sin\( \theta_2 \) cos\( \theta_2 \) ... \( \theta_N \) sin\( \theta_N \) cos\( \theta_N \)

  - sin/cos: >100 cycles v.s. mul/add: 1 cycle
  - Efficient utilization: reduce sin cos operations e.g. sin(a-b)
  - Efficient implementation: alternative version for SIMD datatype

Unrolling for Sparse Kernels

- **General sparse kernel: traversal of the CCS sparse matrix**
  - Original: branch on every non-zero element
  - Unrolled: pre-generate bigger code blocks for multiple columns
  - Bigger code blocks -> better instruction level parallelism

```cpp
for (col = 0; col < n; col++){
    for (row = col_ptr[col]; row < col_ptr[col+1]; row++){
        ...// access & compute on nonzero at (i, j)
    }
}
```

**Unrolling**

```cpp
do{
    switch (case_pattern for 2 consecutive columns){
        case pattern(4, 3): {
            ...// access & compute on nonzero at (i, 1)
            ...// access & compute on nonzero at (i, 2)
            ...// access & compute on nonzero at (i, 3)
            ...// access & compute on nonzero at (i, 4)
            ...// access & compute on nonzero at (i+1, 1)
            ...// access & compute on nonzero at (i+1, 2)
            ...// access & compute on nonzero at (i+1, 3)
            break;
        }
        case ...
    }
}while(!all columns visited)
```
Multilevel Parallelism

- **SIMD**
  - Vector Register: 4 floats in SSE

- **Multithread on multicore**

  | Computing Thd N | SIMD FDPF Load Flow in Buf AN | SIMD FDPF Load Flow in Buf BN |
  | Computing Thd 2 | SIMD FDPF Load Flow in Buf A2 | SIMD FDPF Load Flow in Buf B2 |
  | Computing Thd 1 | SIMD FDPF Load Flow in Buf A1 | SIMD FDPF Load Flow in Buf B1 |
  | Scheduling Thd 0 | KDE in all Buf Bs | Result Out | Sync Signal Out | KDE in all Buf As | Result Out |

**RNG**: Random Number Generator

**KDE**: Kernel Density Estimation

Sync Signal

Real Time Interval (SCADA Interval)
### Result: Performance Breakdown

- **Optimizing iteration speed (on Core i7 2670QM)**

![Performance Graph](image)

#### Approx. Speed: Load Flow Cases Solved per Second on Core i7

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>Baseline</th>
<th>AVX 4-Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Size, Flops/Iteration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1,034</td>
<td>39,000</td>
</tr>
<tr>
<td>24</td>
<td>1,788</td>
<td>23,000</td>
</tr>
<tr>
<td>30</td>
<td>2,242</td>
<td>19,000</td>
</tr>
<tr>
<td>39</td>
<td>2,715</td>
<td>23,000</td>
</tr>
<tr>
<td>57</td>
<td>4,467</td>
<td>15,000</td>
</tr>
<tr>
<td>118</td>
<td>9,130</td>
<td>7,000</td>
</tr>
<tr>
<td>300</td>
<td>23,370</td>
<td>3,000</td>
</tr>
<tr>
<td>2,383</td>
<td>175,365</td>
<td>340</td>
</tr>
</tbody>
</table>

1. **Baseline** is compiler optimized (Intel C Compiler & O3).

- **Upto 60x speedup comparing to scalar baseline (CXSparse)**

- **Converged MCS PLF solution every second**

---

**Core i7 2670QM, quad-core, 2.2GHZ, AVX, 6MB Cache, Gaming Laptop**
**Example on IEEE118 system**

- Normal(0,10MW) and Uniform(-10MW,10MW) random active power plugged into first three highest loading buses (Bus 59: 277MW, Bus 90: 163MW, Bus 116: 184MW)

**Converged MCS PLF solution every second**

(a) Phase angle of Bus 59. Normal distributed P injections

(b) Phase angle of Bus 59. Uniform distributed P injections
Outline: HPC in Substation

- Distribution feeder probabilistic load flow (DPLF)
- Transmission grid probabilistic load flow (TPLF)
- AC contingency calculation (ACCC)
Related Work

Active joint field between power system & computing

- PNNL’s work on supercomputer approach
  - Counter based load balancing\textsuperscript{[PNL_Massive:2009]}
  - Hybrid: Cray-XMT (selection) + cluster (ACCC)\textsuperscript{[PNL_Hyprid:2009]}
- Multicore version ACCC in PTI PSS/E 33 & GE-PSLF SSTOOLS\textsuperscript{[PSSE][PSLF]}
- Drexel’s GPU and FPGA approaches
  - Iterative linear solver on GPU, DC power flow\textsuperscript{[Gopal_GPU:2010]}
  - FPGA accelerated for sparse linear solver\textsuperscript{[Johnson:2008]}

Unique opportunities on commodity CPUs

- Accelerate AC Contingency Calculation by \textit{fully} utilizing CPU hardware?
Our Approach

- **What we have: optimized FDPF computing kernel from TPLF**
  - Efficient linear solver, mismatch computation \( /w \) multilevel parallelism

- **What needed: deal with contingency, topology changes**
  - Goal: fine grain data parallelism for network of different outages?

- **Date level: on single core, SIMD + compensation**
  - Preserves most parts of computations: compensation method
  - Fully utilize the computing power of modern CPU core

- **Task level: on multi-core, thread pool**
  - Convergence at different steps
  - Dynamically balancing workload among cores
Topology Change

- **Handle topology change**
  - Contingency analysis with line trip
  - May change the problem formulation

- **Branch outage representation in admittance matrix:**
  - Given admittance matrix $Y$, branch $i\, j$ change $\Delta y$
    \[
    M = \left[\begin{array}{c}
    0, \ldots, 1, 0 \ldots 0, 0, \ldots, 0
    \\
    0, \ldots, 0, 0 \ldots 0, 1, \ldots, 0
    \end{array}\right]^T
    \Delta y = - \left[\begin{array}{cc}
    y_{ij} + b_{ij} & -y_{ij} \\
    -y_{ij} & y_{ij} + b_{ij}
    \end{array}\right]
    \tilde{Y} = Y + M \Delta y M^T
    \]

- **In FDPF, $B'$ and $B''$ change accordingly**

- **In general, two approaches to handle topology changes**
  - LU re-factorization
  - Compensation method based on Woodbury Identity [Stott_Compensation:1983]

"Sparsity-Oriented Compensation Methods for Modified Network Solutions,"
Solution: Compensation

- **Handle topology change based on compensation**
  - One possible compensation method:
    
    Given: \[ \tilde{A} = A + M \Delta y M^T \]
    
    Goal: Solve \[ \tilde{A} \tilde{x} = b \]
    
    Compute using the lemma:
    
    \[ \tilde{x} = (A^{-1} - A^{-1} M (\Delta y^{-1} + M^T A^{-1} M)^{-1} M^T) A^{-1} b \]
  
  - Steps:
    
    - **Forward:** \( F = L^{-1} b \)
      
    - **Compensate:**
      
      \[ W = L^{-1} M \quad \tilde{W}^T = M^T U^{-1} \]
      
      \[ c = (\Delta y^{-1} + \tilde{W}^T W)^{-1} \]
      
      \[ \Delta F = -W c \tilde{W}^T F \quad \tilde{F} = F + \Delta F \]
      
    - **Backward:** \( \tilde{x} = U^{-1} \tilde{F} \)
Single Core: SIMD Implementation

- Mapping to fine grain parallel hardware

**On FP Unit:**

<table>
<thead>
<tr>
<th>L solve</th>
<th>Compensate</th>
<th>U solve</th>
<th>Compensate</th>
<th>Mismatch</th>
</tr>
</thead>
</table>

**SIMD Unit:**

- In L, U and mismatch: 4 (SSE) or 8 (AVX) cases computed together
- In compensation: serialized and compensate every SIMD slot
- Iteration num determined by the largest iteration num of a SIMD pack

8 (single) or 4 (double) Powerflow Cases of Different Topologies
Single Core Results:

- Scalar speed and AVX speedup on *single* core in Gflop/s

### Speed of ACCC Iterations (Scalar v.s. SIMD)

<table>
<thead>
<tr>
<th>System Size</th>
<th>Speed: Gflop/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>4</td>
</tr>
<tr>
<td>57</td>
<td>4</td>
</tr>
<tr>
<td>118</td>
<td>5</td>
</tr>
<tr>
<td>300</td>
<td>5</td>
</tr>
<tr>
<td>2383</td>
<td>5</td>
</tr>
<tr>
<td>3120</td>
<td>5</td>
</tr>
</tbody>
</table>

- Fully utilize single core with SIMD
- **Better** speedup on *larger* system

Core i7 2670QM, quad-core, 2.2GHZ, AVX, 6MB Cache, Gaming Laptop
Multi-Core: Thread Pool

- Main concern: balancing load among cores
  - Thread pool scheduler
  - Dynamically balancing large amount of small tasks of different convergence steps
Multi-Core Results

- Multi-core speed in Solved Cases / Second

Polish Grid 2383-bus Winter Peak Case

Solved Cases/Second

<table>
<thead>
<tr>
<th>CPU Cores Utilized</th>
<th>1 core</th>
<th>2 cores</th>
<th>3 cores</th>
<th>4 cores</th>
<th>5 cores</th>
<th>6 cores</th>
<th>7 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total: 2252 line outage cases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Polish Grid 3120-bus Summer Peak Case

Solved Cases/Second

<table>
<thead>
<tr>
<th>CPU Cores Utilized</th>
<th>1 core</th>
<th>2 cores</th>
<th>3 cores</th>
<th>4 cores</th>
<th>5 cores</th>
<th>6 cores</th>
<th>7 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total: 2962 line outage cases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Fully utilize computing resources of multi-core CPUs

- Polish Grid N-1 line outage ACCC every second on 4-core Core i7 (~30x speedup over full compiler optimized baseline)

Tolerance: 1kW, Maximal Iteration: 20
Xeon X7560: 8-core, 2.26GHz, SSE, 24MB Cache, Server
Core i7 2670QM, quad-core, 2.2GHZ, AVX, 6MB Cache, Gaming Laptop
Summary

- **DPLF: distribution PLF solver**
  - Millions of load flow cases within SCADA interval
  - Real time accurate, generally-applicable PLF solution
  - Novel full probabilistic SCADA extension

- **TPLF: transmission PLF solver**
  - Optimized solver for fast decoupled power flow
  - Converged accurate MCS PLF results every second for real time apps

- **ACCC: accelerated AC contingency calculation**
  - Fully utilize HW computing capability: cache, instruction level, SIMD, multi-core
  - Complete N-1 for entire national level network (Polish grid) every second

- **HW/SW performance engineering leads to unique solution:**
  Computing enabled situational awareness in substations, control center capability for real time applications targeting critical smart grid challenges
HW/SW performance engineering leads to unique solution:
Computing enabled situational awareness in substations, control center capability for real time applications targeting critical smart grid challenges
References

The End

Thank you!

Q&A