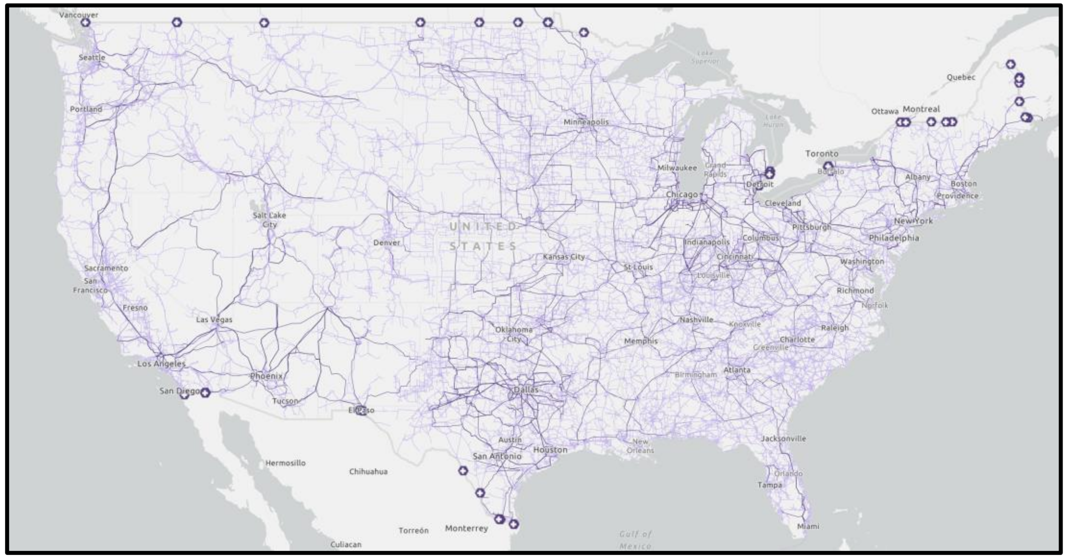
The Pursuit for a Generalized Simulation Framework for Power System Transients and Stability

Hantao Cui, Associate Professor
Electrical and Computer Engineering
NC State University

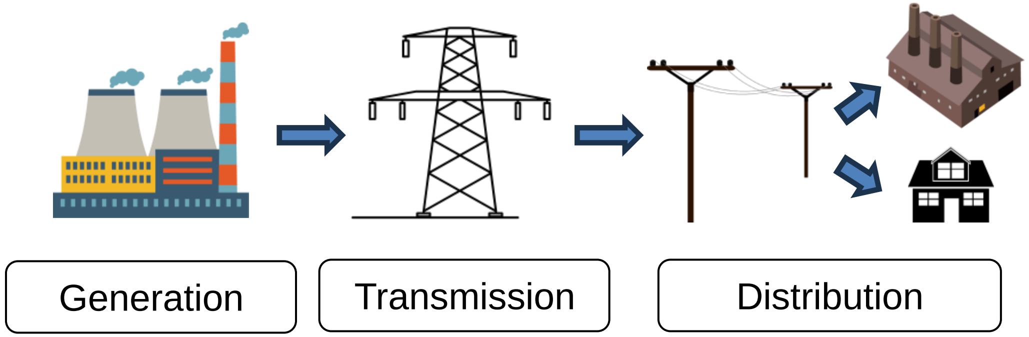
IEEE Young Professionals Webinar, Phoenix Section
Wednesday, October 30, 2024

Transmission Grid as a Critical Infrastructure

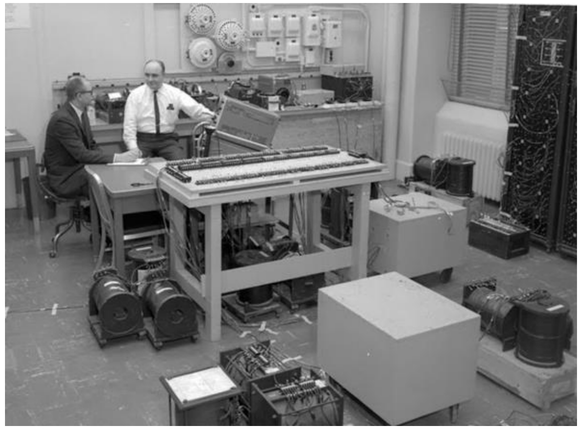
- Electric power grids are part of the critical infrastructure for energy delivery
- Requires **instantaneous balancing** of power across a synchronized network to maintain secure operation
- There is always a need to understand grid stability and security



Continental U.S. | Transmission Grids



A Long History of Power System Simulation



TRANSIENT NETWORK ANALYZER (TAN)

Made of RLC circuits, scaled down generators and motors

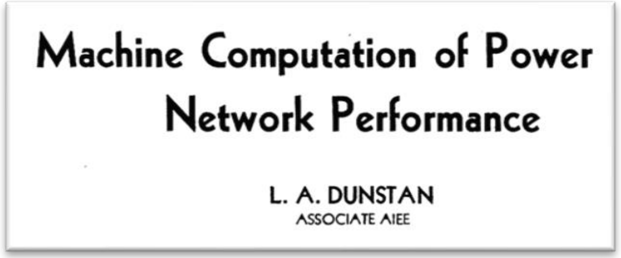
Popular from 1929 to the late 1960s

ENIAC UNVEILED AT UPENN

The widely recognized first programmable, electric, general-purpose computer was completed at UPenn.

1945

1948



Machine Computation of Power Network Performance

L. A. DUNSTAN
ASSOCIATE AIEE

POWER FLOW FORMULATION

Power flow formulation was formalized

Late 1940s

1957

Harry M. Markowitz

SPARSE MATRIX TECHNIQUES

Markowitz and collaborators developed sparse matrix methods while studying portfolio selection theory from 1952 to 1957.

He won the Nobel Prize in Economics in 1990

A Long History of Power System Simulation

1963


Techniques for Exploiting the Sparsity of the Network Admittance Matrix

NOBUO SATO
MEMBER IEEE

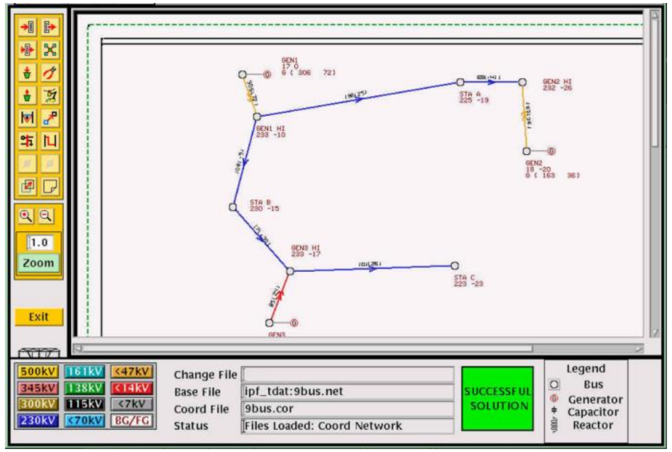
W. F. TINNEY
MEMBER IEEE

SPARSE MATRIX APPLIED TO POWER FLOW

Sato and Tinney employed sparse matrix methods to solve power flow.

Pivoting techniques developed by Tinney for power applications are widely used today.

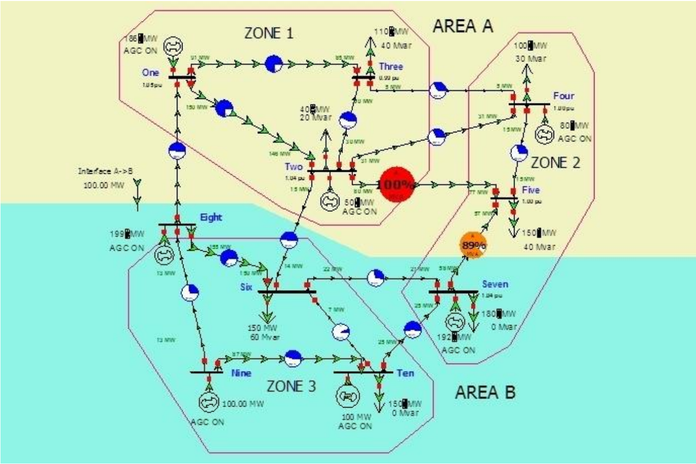
1960s – 1980s



SOFTWARE AND GRAPHICAL USER INTERFACES

Many tools have been developed, such as BPA and EMTP. Some of them have a graphical user interface for usability

1990s – Today



SOFTWARE AND METHODS CONTINUE TO EVOLVE

New computer hardware necessitates new algorithms and tools.

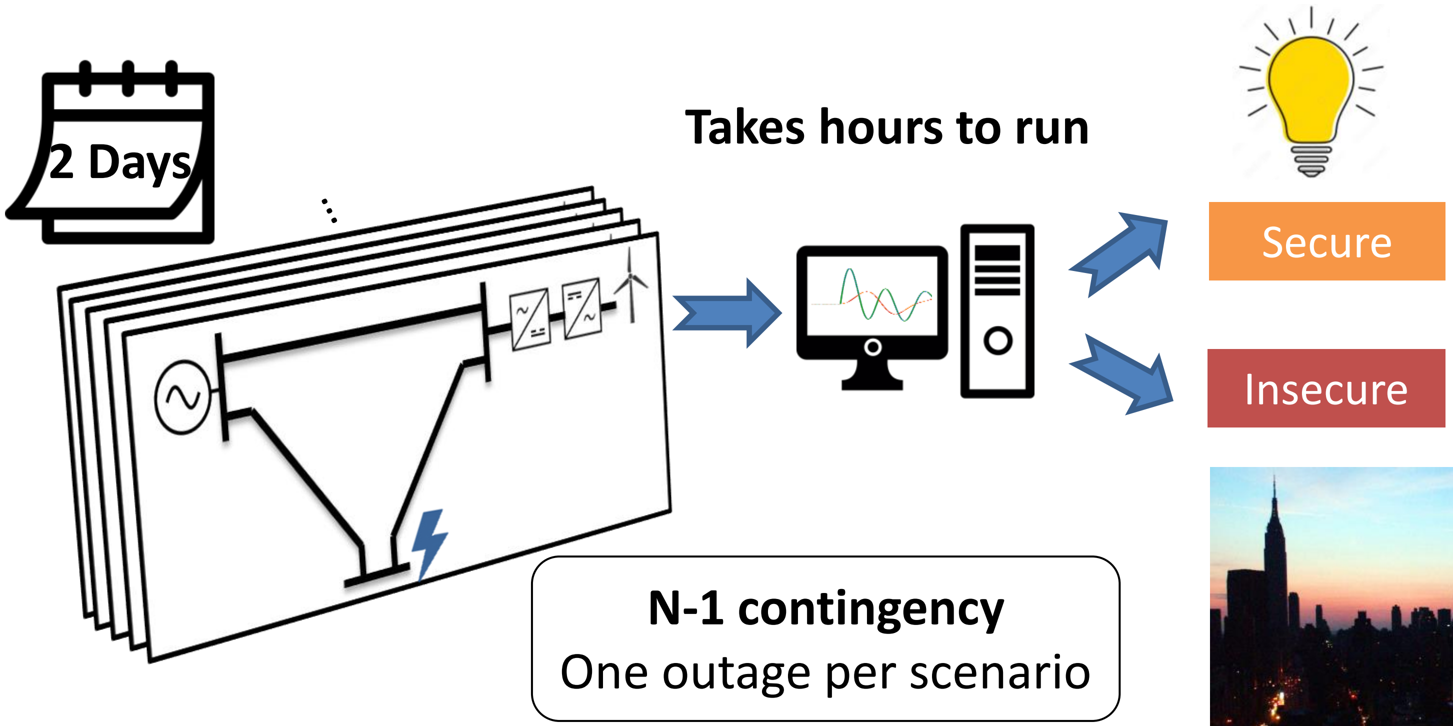
This evolution is coupled with the rapid integration of renewable energy

Grid Security Planning of Today: Modeling and Simulation

System security is subject to disturbances. What-if…

- Line trip? Generator outage? Wind gust? Solar eclipse?

Answers obtained by computer simulation of grid models



Meaningful results depend on:

- Model **accuracy**
- Simulation **timeliness**

Grid Security Planning of Today: Modeling and Simulation

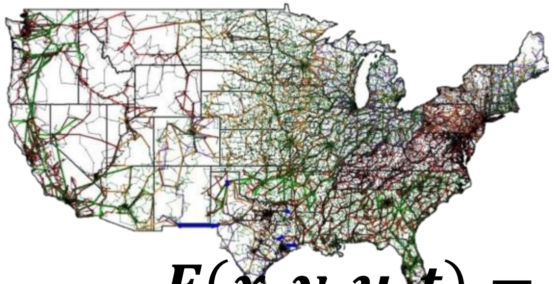
Models characterize dynamic system behaviors using mathematical equations

Devices

$$\begin{bmatrix}\dot{x}_{LG}\\ \dot{x}_{LL}\end{bmatrix} = \begin{bmatrix} z_{i,lim}^{LG}\left(P_d - x_{LG}\right)/T_1 \\ \left(x_{LG} - x_{LL}\right)/T_3 \end{bmatrix}$$

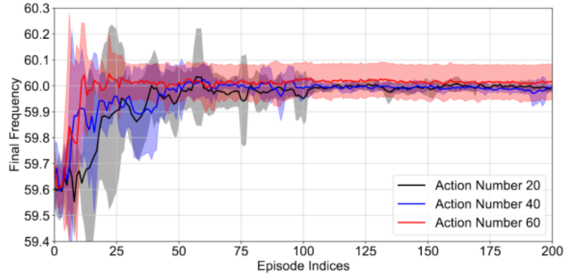
$$\begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix} (1-\omega) - \omega_d \\ R \times \tau_{m0} - P_{ref} \\ \left(P_{ref} + \omega_d\right)/R - P_d \\ D_t \omega_d + y_{LL} - P_{OUT} \\ \frac{T_2}{T_3}\left(x_{LG} - x_{LL}\right) + x_{LL} - y_{LL} \\ u\left(P_{OUT} - \tau_{m0}\right) \end{bmatrix}$$

Device models

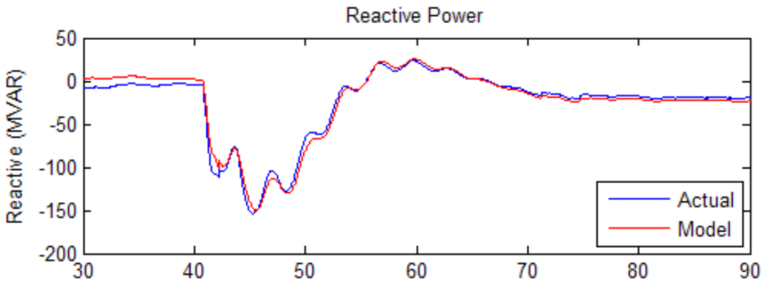


$$\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{t}) = 0$$

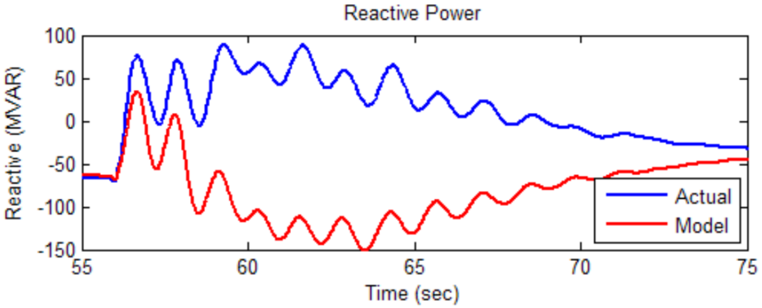
Grid models



Simulation results



A good model makes predictions



A bad model hides issues

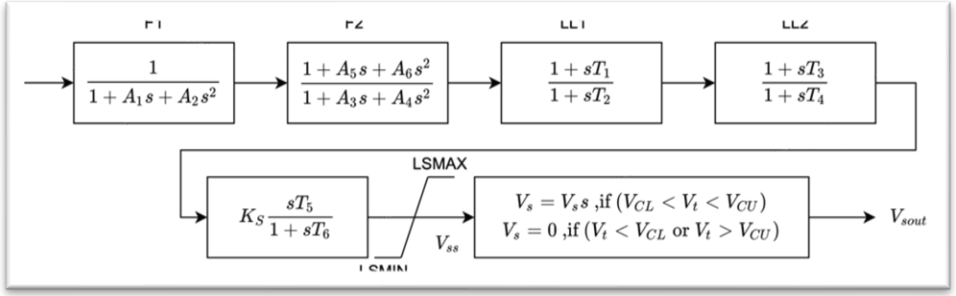
Challenge 1/2: Complexity in Modeling Renewables

Grid security is predictable… **until recently**

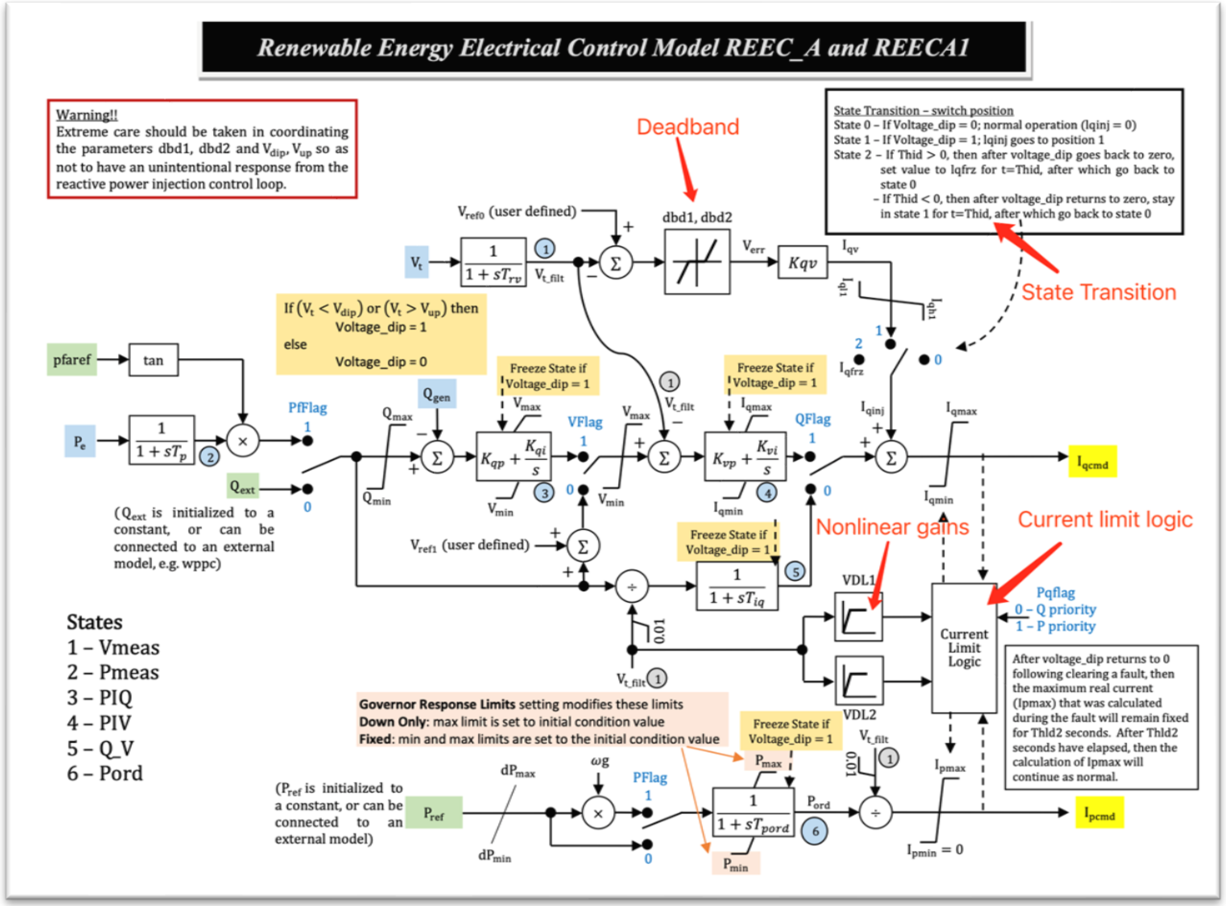
<u>Renewables</u> greatly contribute to sustainability, but

- Significantly more **complex**
- **Non-conventional** logic

Challenges in correctness and accuracy due to **complexity**



IEEEST widely used for large-scale systems



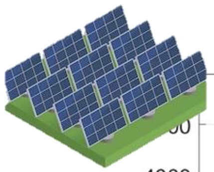
Renewable energy control model. (Credit: Powerworld)

Challenge 2/2: Simulation of Renewable-Dominated Systems

Computational challenges due to renewables

- Internal model complexity -> more computational burden per device
- High uncertainty necessitates more scenarios to cover low-probability events

Computational complexity necessitates **high-performance methods** for renewable-dominated grids



Rapidly variable solar power

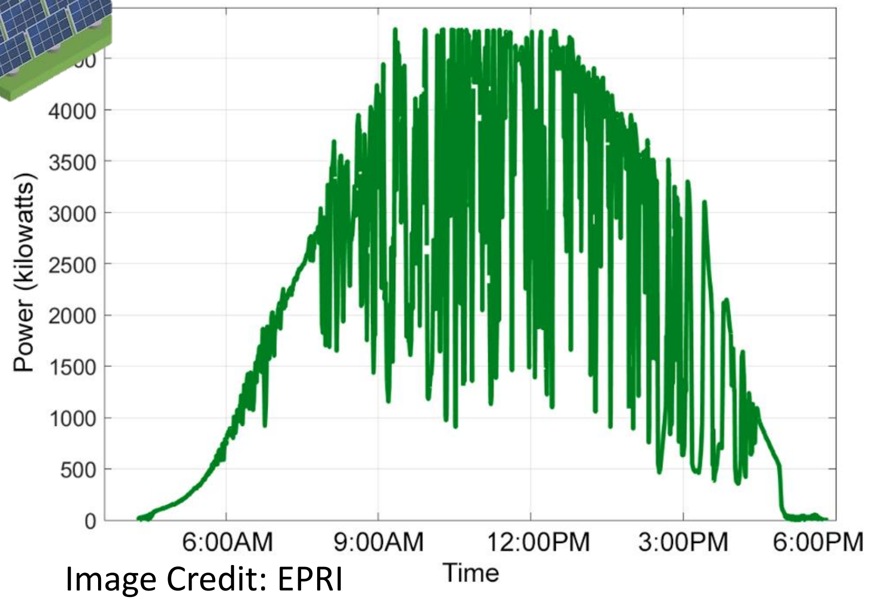
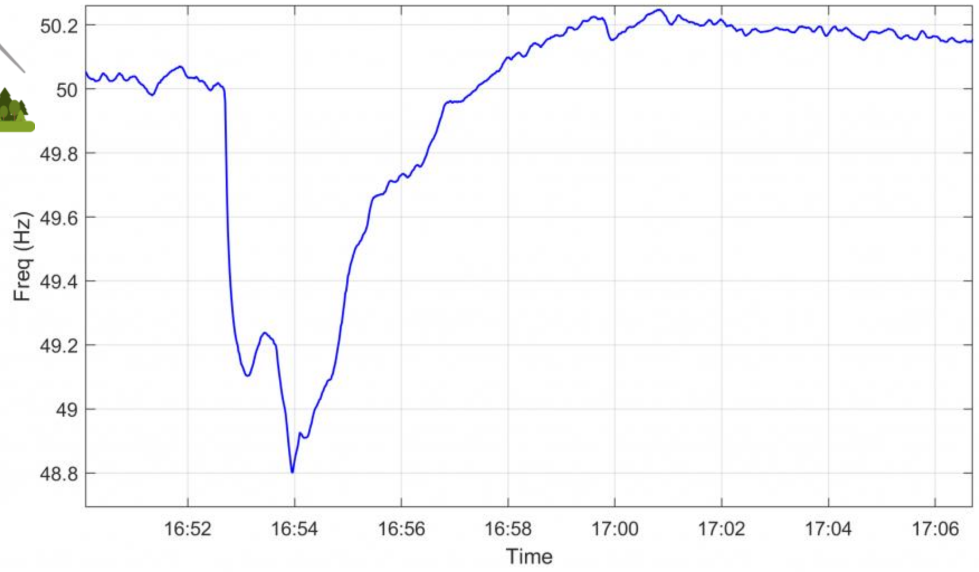


Image Credit: EPRI

GB National Grid Event on August 9, 2019

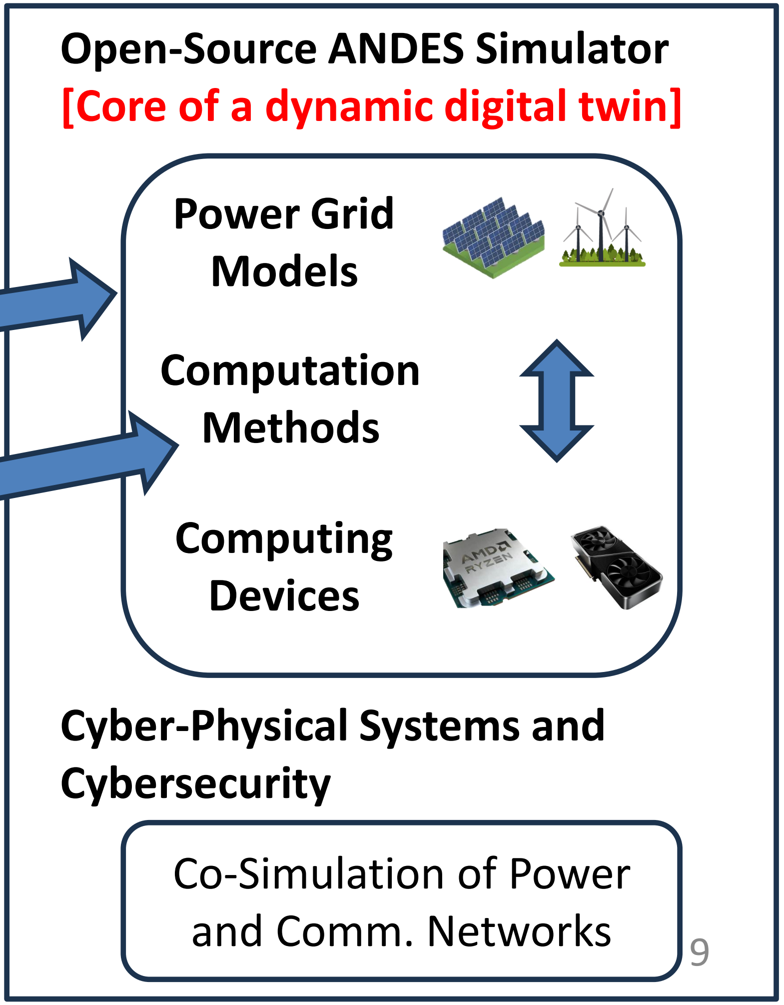


Research Interest and Core Expertise

Research Interest: Empowering **sustainability transition** by computational and emerging technologies for secure, renewable-friendly, and efficient power grids.

Present Challenges	*My Solutions*
Complexity in Modeling	A new **paradigm** for modeling and simulation
Computation speed challenge	Reformulate power problems for **parallelization**

- An **interdisciplinary ecosystem** for research
- Accelerate translation of research into applications



Outline of this Talk

1. **A hybrid symbolic-numeric framework** for descriptive modeling and fast simulation
2. **An element-wise approach** for power flow calculation alternative to admittance matrix
3. **Ongoing studies** to unifying modeling and accelerating computation

Current State of Grid Dynamics Modeling: Complexity

Need to formulate and **implement device models** before simulation

Ad hoc Implementation

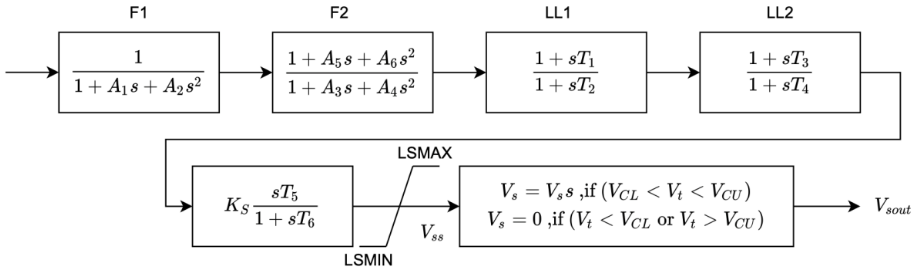
- Steady learning curve
- Error-prone; difficult to scale to large systems

Commercial Tools

- Large model libraries; good computation speed
- Black box; difficult to customize

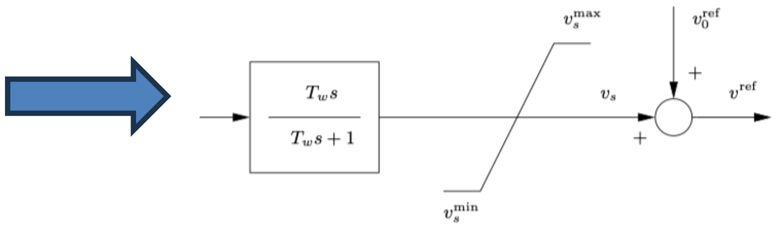
Open-Source Tools

- Models and algorithms can be readily modified for research
- (Over)-simplification that compromises accuracy



IEEEST widely used for large-scale systems

```
def gcall(self, dae):
    dae.g[self.In] = mul(
        self.u0,
        -dae.y[self.In] + mul(self.Ic1, -1 + dae.x[self.omega]) + mul(
            self.Ic2, -1 + dae.x[self.w]) + mul(self.Ic5, dae.y[self.v]) +
        mul(self.Ic3, dae.y[self.p], self.toSg) + mul(
            self.Ic4, dae.y[self.pm], self.toSg))
    dae.g[self.v1] = mul(
        self.u0, dae.x[self.q0] - dae.y[self.v1] + mul(
            self.A5, dae.x[self.q1]) + mul(self.A6, dae.x[self.q2]))
    dae.g[self.v2] = mul(
        self.u0,
        dae.x[self.x1] - dae.y[self.v2] + mul(self.T12, dae.y[self.v1]))
```



Current State of Grid Dynamics Modeling: Complexity

Need to formulate and **implement device models** before simulation

Ad hoc Implementation

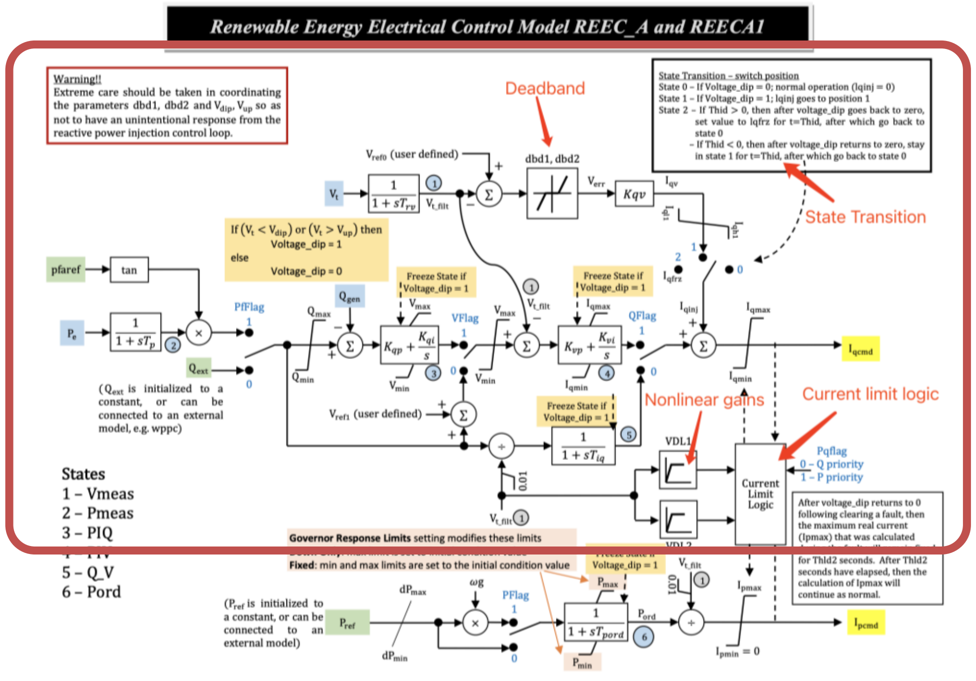
- Steady learning curve
- Error-prone; difficult to scale to large systems

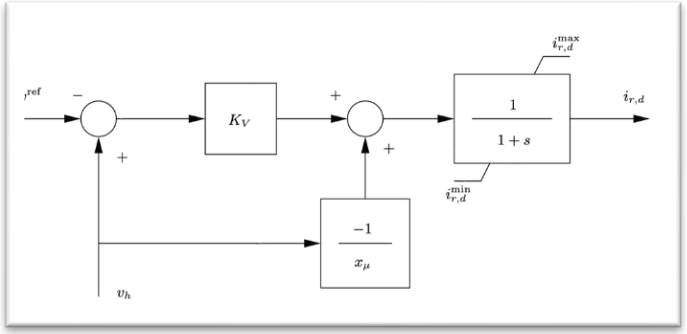
Commercial Tools

- Large model libraries; good computation speed
- Black box; difficult to customize

Open-Source Tools

- Models and algorithms can be readily modified for research
- (Over)-simplification that compromises accuracy

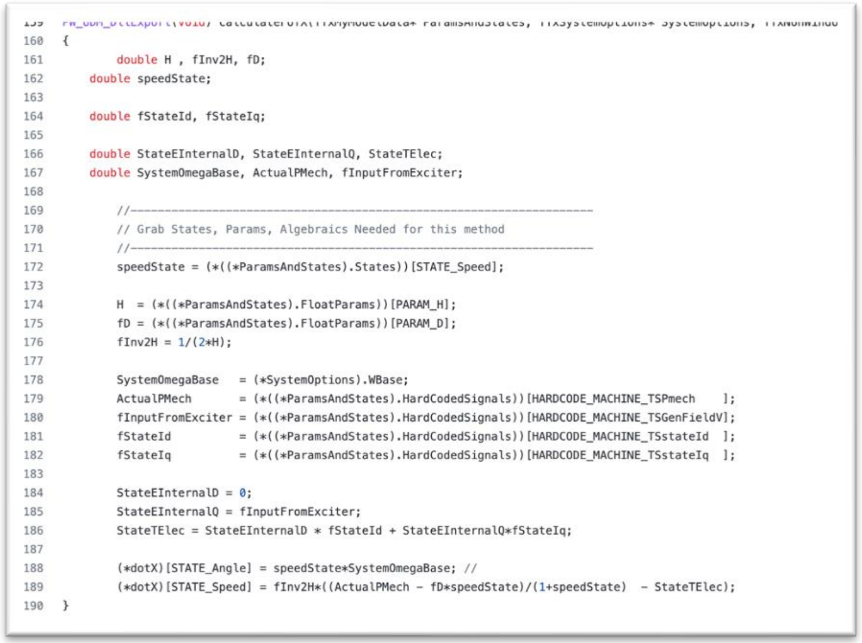




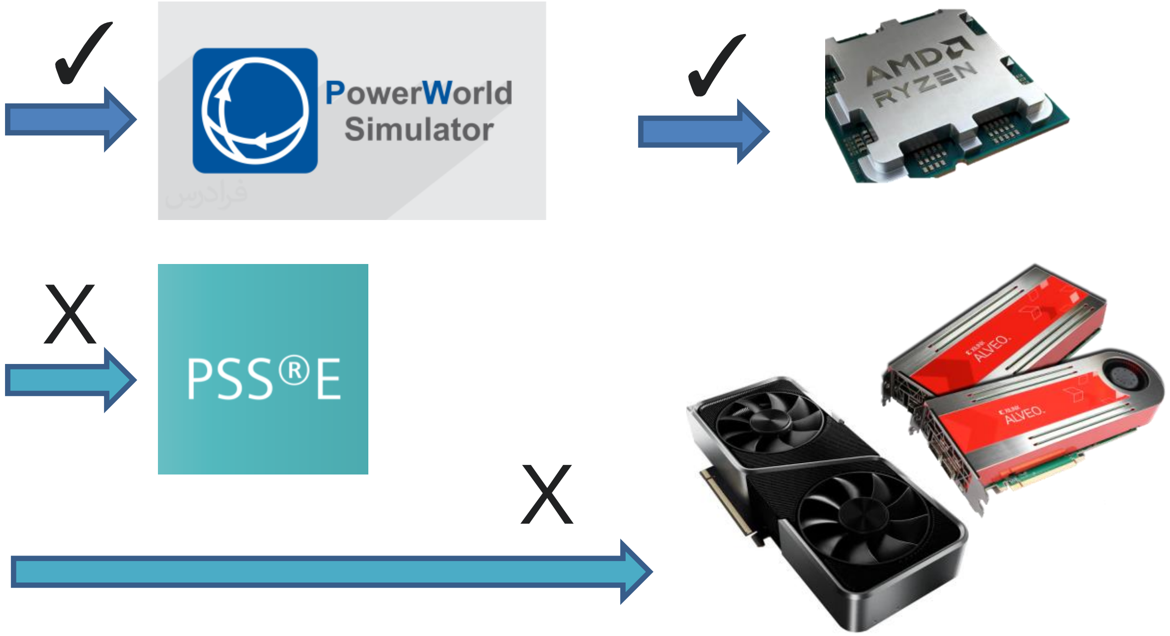
Current State of Modeling: Interoperability and Performance

The issue with model "**implementation**"

- One implementation works only in one framework, lacking interoperability
- Without **reimplementation**, legacy frameworks can hardly leverage new computing capabilities



```
{
        double H , fInv2H, fD;
    double speedState;

    double fStateId, fStateIq;

    double StateEInternalD, StateEInternalQ, StateTElec;
    double SystemOmegaBase, ActualPMech, fInputFromExciter;

        //----------------------------------------------------------------
        // Grab States, Params, Algebraics Needed for this method
        //----------------------------------------------------------------
        speedState = (*((*ParamsAndStates).States))[STATE_Speed];

        H  = (*((*ParamsAndStates).FloatParams))[PARAM_H];
        fD = (*((*ParamsAndStates).FloatParams))[PARAM_D];
        fInv2H = 1/(2*H);

        SystemOmegaBase   = (*SystemOptions).WBase;
        ActualPMech       = (*((*ParamsAndStates).HardCodedSignals))[HARDCODE_MACHINE_TSPmech    ];
        fInputFromExciter = (*((*ParamsAndStates).HardCodedSignals))[HARDCODE_MACHINE_TSGenFieldV];
        fStateId          = (*((*ParamsAndStates).HardCodedSignals))[HARDCODE_MACHINE_TSstateId  ];
        fStateIq          = (*((*ParamsAndStates).HardCodedSignals))[HARDCODE_MACHINE_TSstateIq  ];

        StateEInternalD = 0;
        StateEInternalQ = fInputFromExciter;
        StateTElec = StateEInternalD * fStateId + StateEInternalQ*fStateIq;

        (*dotX)[STATE_Angle] = speedState*SystemOmegaBase; //
        (*dotX)[STATE_Speed] = fInv2H*((ActualPMech - fD*speedState)/(1+speedState)  - StateTElec);
}
```



Call for a **paradigm shift** in modeling to leverage new capabilities

Computer-Assisted Symbolic Approach to Grid Modeling

Guiding Principle:

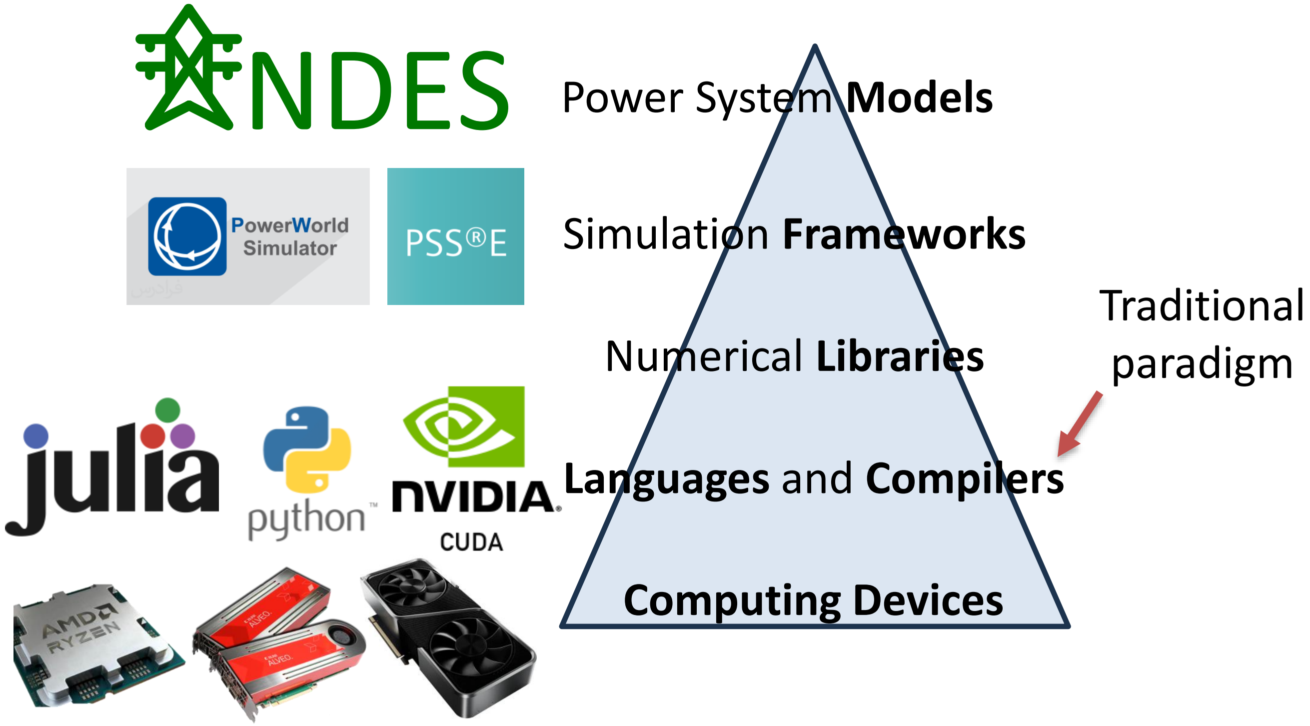
- Describe models ***just for once*** and reuse them in different ways

How? By leveraging existing scientific computing infrastructure.

Objectives:

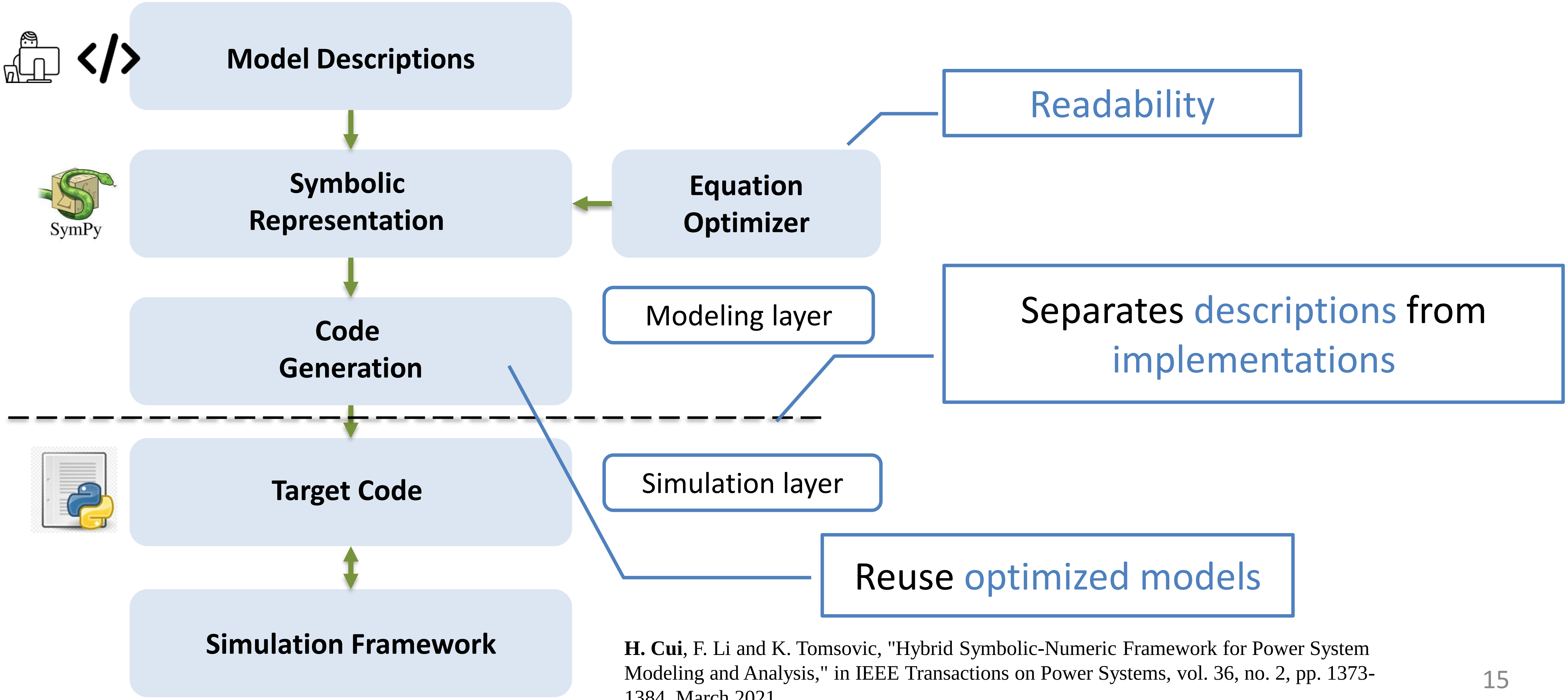
- **Accuracy** in modeling and simulation
- Computational **efficiency**
- **Productivity** and **interoperability**

Significance: automatically harvest new capabilities from computing domain.



Perspective of software-hardware stack for power system computing

ANDES Simulator for Descriptive Modeling: Architecture



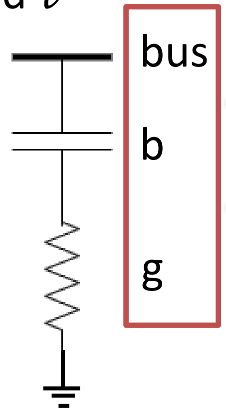
H. Cui, F. Li and K. Tomsovic, "Hybrid Symbolic-Numeric Framework for Power System Modeling and Analysis," in IEEE Transactions on Power Systems, vol. 36, no. 2, pp. 1373-1384, March 2021.

Descriptive Modeling using Equations

- **Equation-based** generalized modeling

θ and v

Variables



Shunt compensator

$$p_h = g v_h^2$$
$$q_h = -b v_h^2$$

Parameters

Variables

```
class Shunt(Model):
  def __init__(self):
    self.bus = IdxParam(info='bus_index')
    self.g = NumParam(info='conductance', unit='pu')
    self.b = NumParam(info='susceptance', unit='pu')
    self.a = ExtAlgeb(model='Bus', indexer=self.bus,
                      src='a', e_str='g*v*v')
    self.v = ExtAlgeb(model='Bus', indexer=self.bus,
                      src='v', e_str='-b*v*v')
```

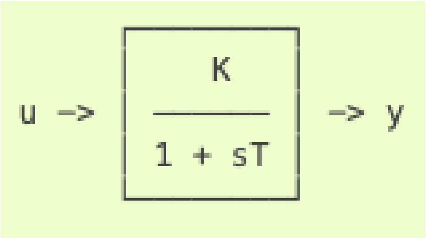
Equations

```
@numba.jit
def g_update(g, v, b):
      return ((g*v**2, -b*v**2))

def gy_update(g, v, b):
      return ((2*g*v, -2*b*v))
```

Automatic dx/dt

Descriptive Modeling using Blocks

A low-pass filter (lag)



$$T\dot{y}_{LG} = Ku - y_{LG}$$

Block-based modeling (✓)

```
self.LG = Lag(u=self.u, T=self.T, K=1)
```

Automatic

```
self.LG_y = State(e_str='u - LG_y', t_const=self.T,
                  v_str='u')
```

control blocks:

- Lag, LeadLag, Washout, Gain, Integrator with respective anti-windup variants
- PID controllers with various anti-windup limiters

discontinuities:

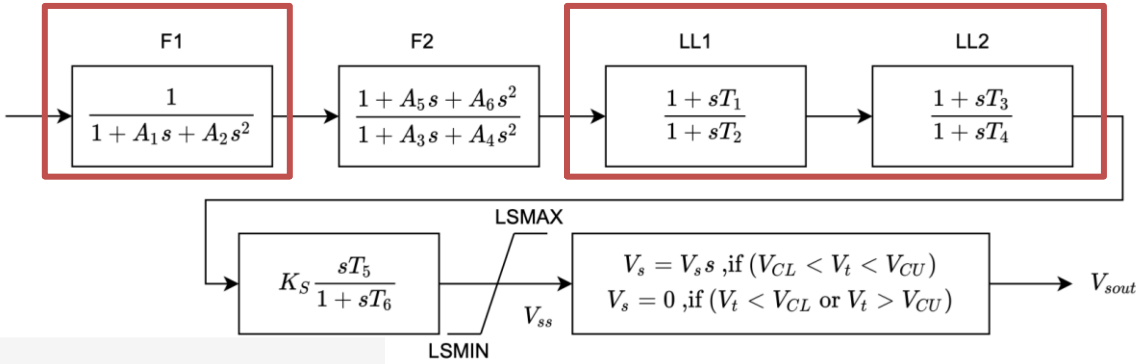
- HardLimiter, Antiwindup, SortedLimiter
- Deadbands
- RateLimiter, AntiwindupRate
- Delay, Average, Derivative, Sampling

services (helpers):

- ConstService, VarService, RandomService
- VarHold, EventFlag
- FlagCondition
- …

- Can construct ~1,000 models
- Complexity scale to # of blocks

Symbolics-Numeric Modeling Framework: An Example

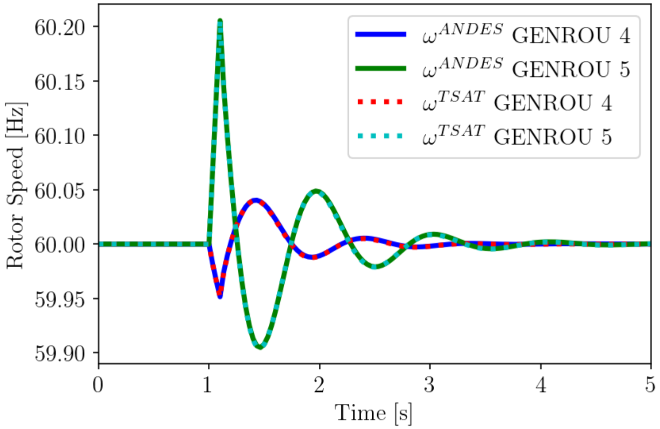


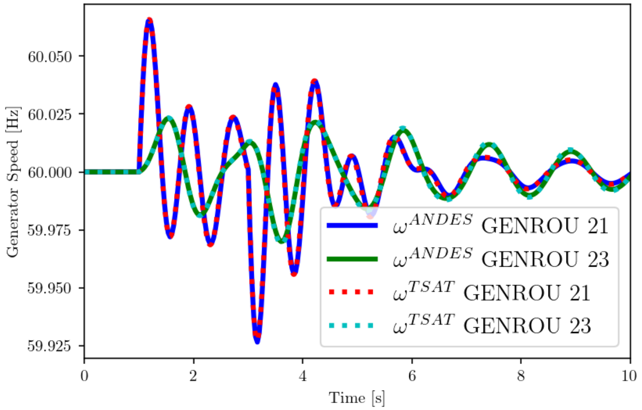
Python code snippet for describing IEEEST

```
from andes.core import Model, Limiter
from andes.core import Lag2ndOrd, LeadLag2ndOrd, LeadLag, WashoutOrLag, Gain


class IEEESTModel(Model):
    def __init__(self):
        Model.__init__(self)
        self.F1 = Lag2ndOrd(u=self.sig, K=1, T1=self.A1, T2=self.A2)
        self.F2 = LeadLag2ndOrd(u=self.F1_y,
                                T1=self.A3, T2=self.A4, T3=self.A5, T4=self.A6)

        self.LL1 = LeadLag(u=self.F2_y, T1=self.T1, T2=self.T2)
        self.LL2 = LeadLag(u=self.LL1_y, T1=self.T3, T2=self.T4)
        self.Vks = Gain(u=self.LL2_y, K=self.KS)
        self.WO = WashoutOrLag(u=self.Vks_y, T=self.T6, K=self.T5, name='WO')

        self.VLIM = Limiter(u=self.WO_y, lower=self.LSMIN, upper=self.LSMAX, info='Vss limiter')
        self.Vss = Algeb(e_str='VLIM_zi * WO_y + VLIM_zu * LSMAX + VLIM_zl * LSMIN - Vss')

        self.OLIM = Limiter(u=self.v, lower=self.VCLr, upper=self.VCUr, info='output limiter')
        self.vsout.e_str = 'OLIM_zi * Vss - vsout'

```

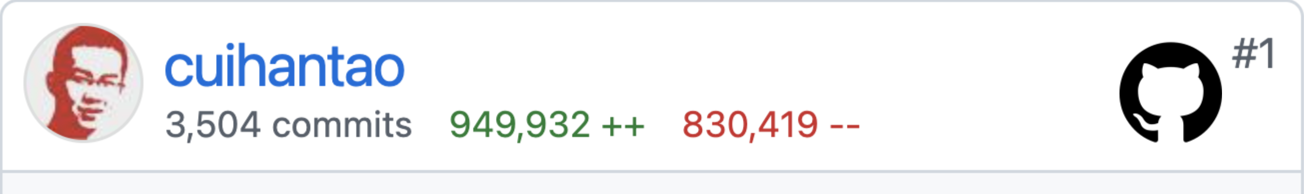
✓ Productivity for researchers
✓ Maintainable

The CURENT ANDES Simulator: A Full-Fledged Package

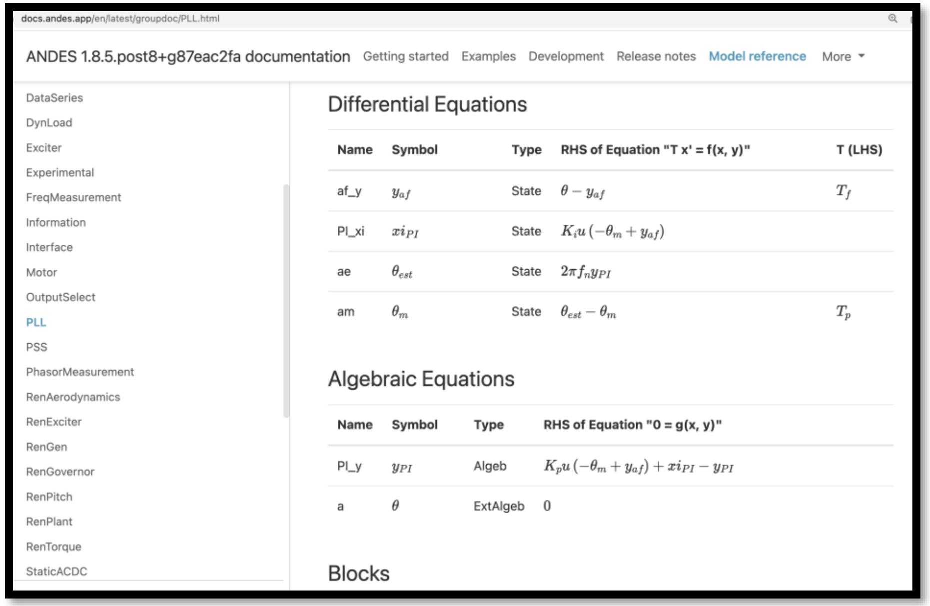




Matching results with TSAT using IEEE 14-bus and NPCC 140 systems



- Provides ~100 models
- Power flow methods
- Transient stability & small-signal stability analysis
- **Interoperates** with MATPOWER and pandapower for optimal power flow
- Used for control, data analytics and machine learning



Quality documentation enables collaborations:

- Binghamton, Mines, UPC in Spain
- Researchers who develop open-source tools

Outline of this Talk

1. **A hybrid symbolic-numeric framework** for descriptive modeling and fast simulation
2. **An element-wise approach** for power flow calculation alternative to admittance matrix
3. **Ongoing studies** to unifying modeling and accelerating computation

Fine-Grained Parallelization for Accelerated Computation

Motivation: Accelerate the simulation of large-scale systems

Power system **simulation** = **solving** large-scale differential algebraic equations

Two inherently serial steps:

1. Calculate residuals and Jacobians
2. Solve linear eqns. (efficient libraries)

$$\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{u},\boldsymbol{t}) = 0$$

Calculate Residuals and Jacobians

$$\boldsymbol{res} = \boldsymbol{F}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{u},\boldsymbol{t})\Big|_{x,y,u}$$

$$\boldsymbol{J} = d\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{u},\boldsymbol{t})/d(x,y)$$

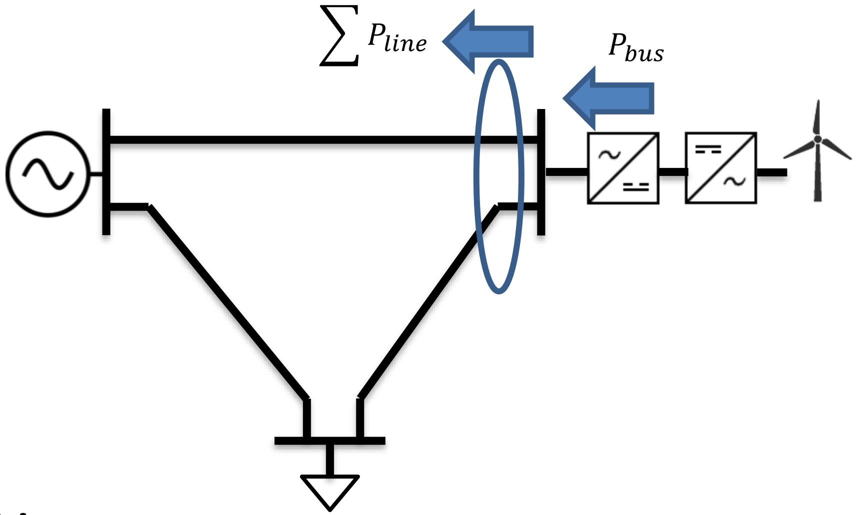
Solve linear equations in iterations

$$\boldsymbol{\Delta x} = -\mathbf{J}\backslash\mathrm{F}$$

```
ncalls  cumtime  percall filename:lineno(function)
     1    1.828    1.828 pflow.py:155(nr_solve)
     9    0.852    0.095 linsolvers/suitesparse.py:93(solve)
     9    0.820    0.091 system.py:1072(j_update)
```

Jacobian calculate is a major computation effort

Existing Approach: Bus Admittance Matrix



Aim:

- calculate $\sum P_{line}$ for each bus

Computational Challenges:

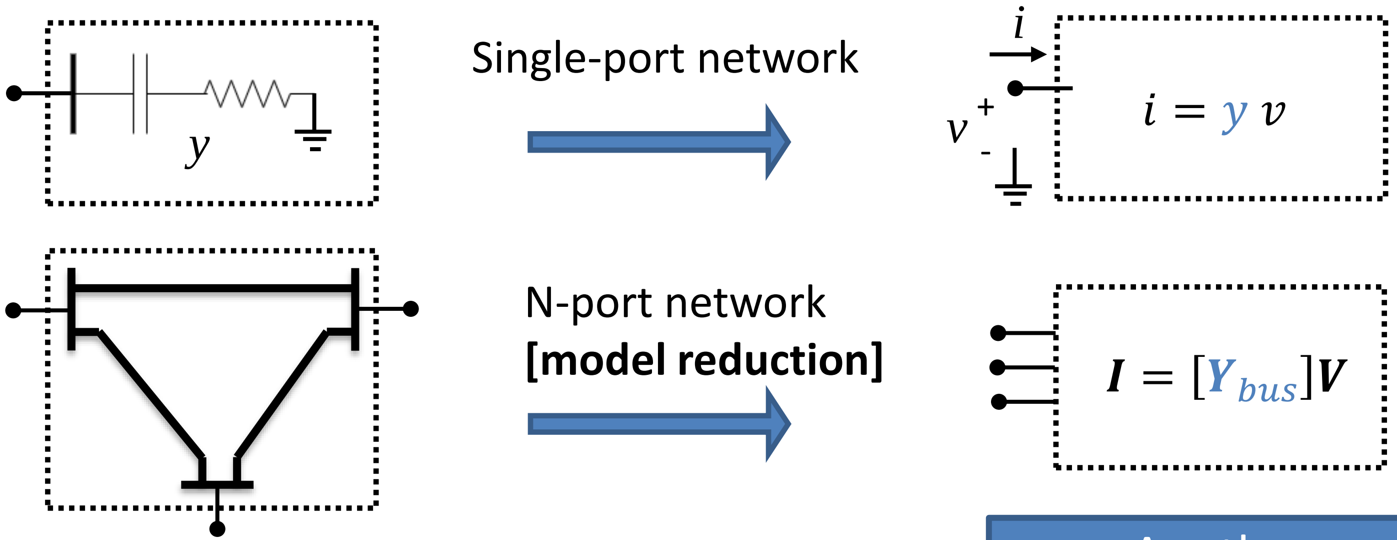
- Four power equations per line
- many lines (>150,000)

Opportunity:

- fewer buses (<80,000)

Can we calculate it by bus?

Admittance matrix method—textbook method



Complex power injections:

$$S = P + jQ = VI^* = V([Y_{bus}]\,V)^*$$

Jacobian matrix:

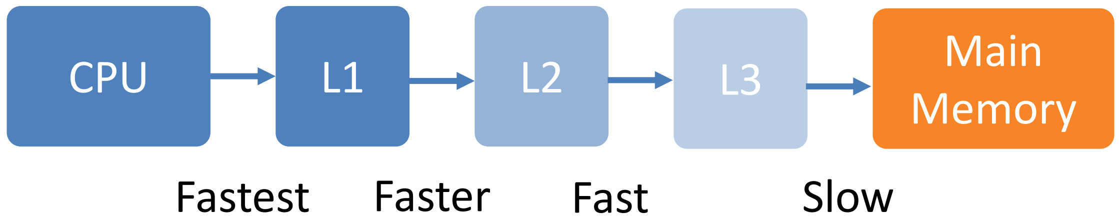
$$J = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} = \begin{bmatrix} \mathrm{Re}(\partial S/\partial \theta) & \mathrm{Re}(\partial S/\partial V) \\ \mathrm{Im}(\partial S/\partial \theta) & \mathrm{Im}(\partial S/\partial V) \end{bmatrix}$$

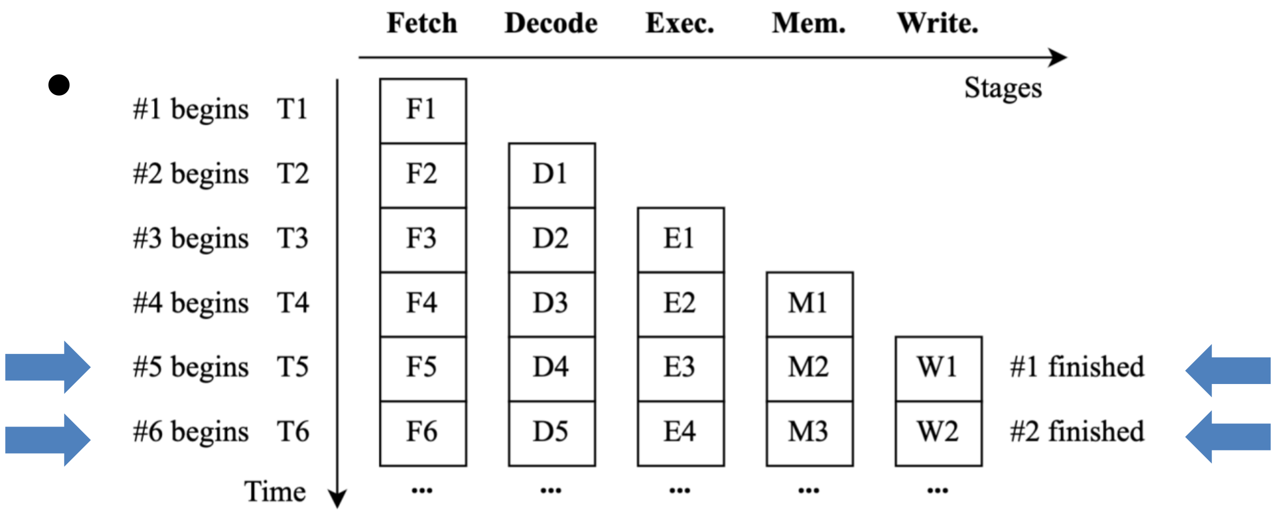
Are these computation efficient on *modern* CPUs?

Computing the Jacobian Matrix using Admittance Matrix

Modern CPU features

- Multiple levels of cache





Dynamic indexing

More cache misses!

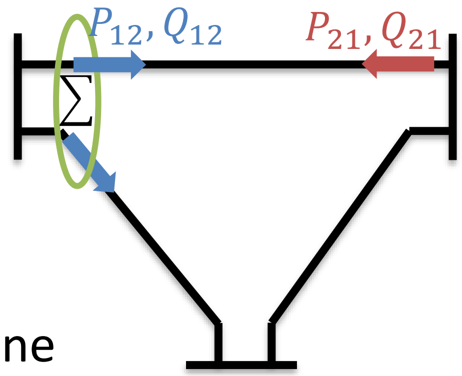
Jump

Stalls pipelines!

Algorithm 1 Two-pass method for Jacobian using CSC

INPUT: $\boldsymbol{Y}_p$, $\boldsymbol{Y}_i$, $\boldsymbol{Y}_v$, $d\boldsymbol{S}_\theta$, $d\boldsymbol{S}_V$, $\mathbf{I}_{\text{bus}}$, $\boldsymbol{V}$, $\boldsymbol{U}$
INITIALIZE:
    $\boldsymbol{I}_{bus} = 0$, $d\boldsymbol{S}_\theta.\text{nzval} .= \boldsymbol{Y}_v$, $d\boldsymbol{S}_V.\text{nzval} .= \boldsymbol{Y}_v$
PASS 1: for $j = 1 : n_b$ (j is column index)
    for $k = \boldsymbol{Y}_p[j] : (\boldsymbol{Y}_p[j+1] - 1)$ (k is j's index range)
        $\boldsymbol{I}_{bus}[\boldsymbol{Y}_i[k]] \mathrel{+}= \boldsymbol{Y}_v[k] * \boldsymbol{V}[j]$
        $d\boldsymbol{S}_\theta.\text{nzval}[k] \times= \boldsymbol{V}[j]$
        $d\boldsymbol{S}_V.\text{nzval}[k] \times= \boldsymbol{U}[j]$
    end for k
end for j
PASS 2: for $j = 1 : n_b$
    for $k = \boldsymbol{Y}_p[j] : (\boldsymbol{Y}_p[j+1] - 1)$
        $i = \boldsymbol{Y}_i[k]$ (i is element k's row number)
        $d\boldsymbol{S}_V.\text{nzval}[k] = \boldsymbol{V}[i] \times (d\boldsymbol{S}_V[K])^*$
        if $i == j$
            $d\boldsymbol{S}_\theta.\text{nzval}[k] \mathrel{-}= \boldsymbol{I}_{\text{bus}}[j]$
            $d\boldsymbol{S}_V.\text{nzval}[k] \mathrel{+}= (\boldsymbol{I}_{bus}[j])^* \times \boldsymbol{U}[j]$
        end if
        $d\boldsymbol{S}_\theta.\text{nzval}[k] = (1im) \times (d\boldsymbol{S}_\theta.\text{nzval}[k])^* \times \boldsymbol{V}[i]$
    end for k
end for j
RETURN: $d\boldsymbol{S}_\theta$, $d\boldsymbol{S}_V$

Proposed Approach: Line Element-wise Calculation



Step 1: injections per line

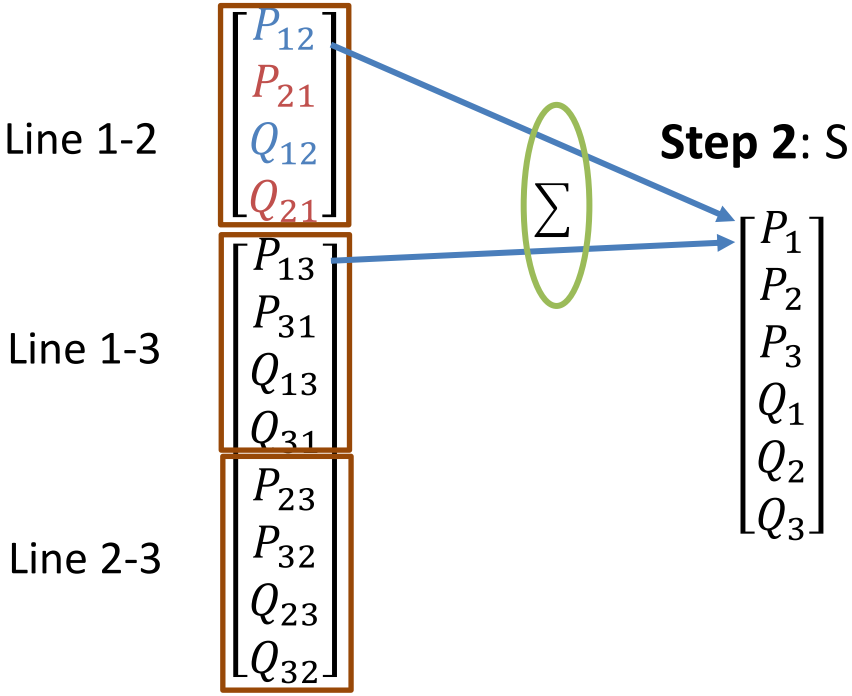
Evaluate four equations per line

$$P_{hk} = Re[-v_h v_k \frac{y_{hk}^*}{m} e^{-j(\theta_h-\theta_k-\phi)} + v_k^2 (y_k + y_{hk})^*]$$

$$Q_{hk} = Im[-v_h v_k \frac{y_{hk}^*}{m} e^{-j(\theta_h-\theta_k-\phi)} + v_k^2 (y_k + y_{hk})^*]$$

$$P_{kh} = Re[\ v_h^2 \frac{(y_h + y_{hk})^*}{m^2} - v_h v_k \frac{y_{hk}^*}{m} e^{j(\theta_h-\theta_k-\phi)}\ ]$$

$$Q_{kh} = Im[\ v_h^2 \frac{(y_h + y_{hk})^*}{m^2} - v_h v_k \frac{y_{hk}^*}{m} e^{j(\theta_h-\theta_k-\phi)}\ ]$$



Step 2: Sum up the injections

Step 1: **Map** data to equations

Step 2: **Reduce** outputs by summation

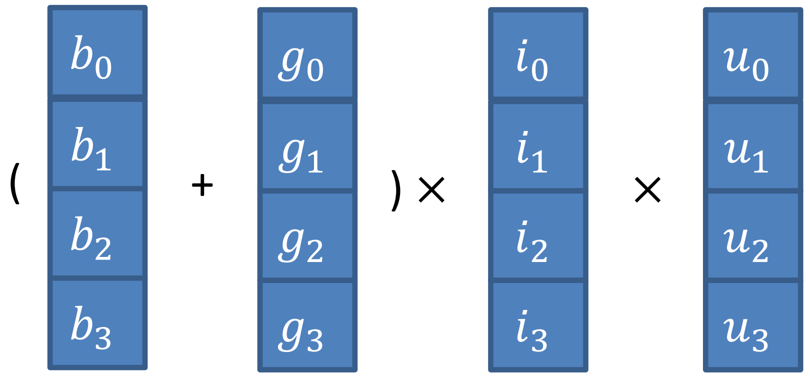
- Scales to n_{lines}
- How can this method be faster than Y_{bus} method?

Line Element-wise Calculation for Data Parallelism

Julia code generated by ANDES

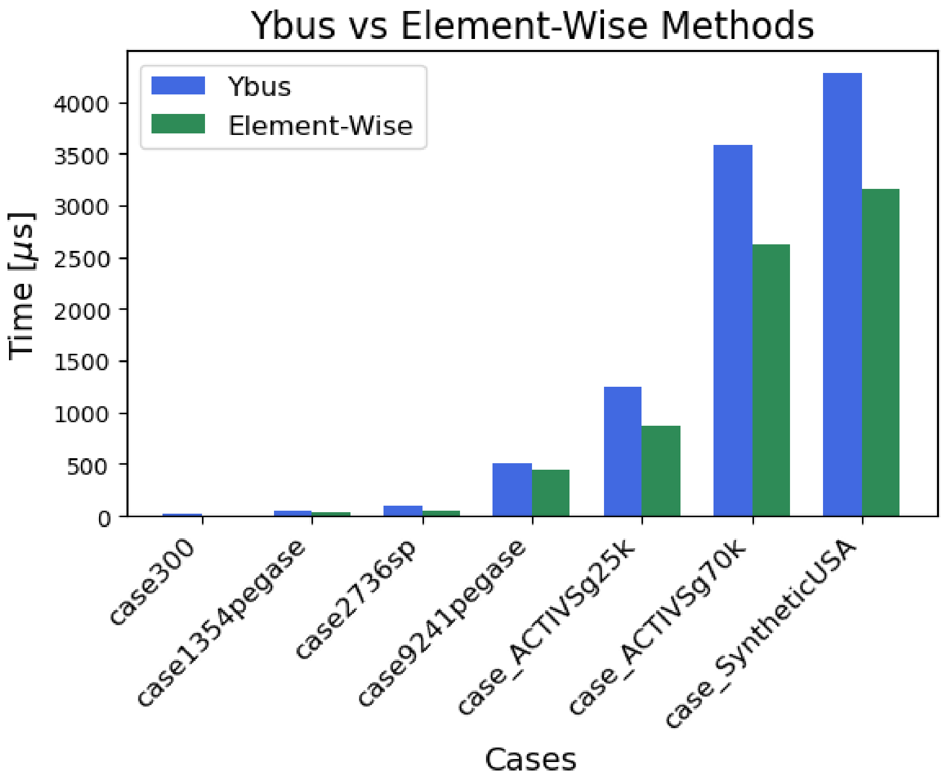
```julia
    @turbo for m in eachindex(u)
        _gy1[m] = -u[m] * itapv1v2[m] * (bhkcosine[m] + ghksine[m])
        _gy2[m] = -u[m] * itapv1v2[m] * (-bhkcosine[m] - ghksine[m])
        _gy3[m] =
            u[m] *
            (-itapv2[m] * (-bhksine[m] + ghkcosine[m]) + 2 * v1[m] * itap2_yhyhkconj.re[m])
        _gy4[m] = -u[m] * itapv1[m] * (-bhksine[m] + ghkcosine[m])
        _gy5[m] = -u[m] * itapv1v2[m] * (-bhkcosine[m] + ghksine[m])
        _gy6[m] = -u[m] * itapv1v2[m] * (bhkcosine[m] - ghksine[m])
        _gy7[m] = -u[m] * itapv2[m] * (bhksine[m] + ghkcosine[m])
        _gy8[m] =
            u[m] * (-itapv1[m] * (bhksine[m] + ghkcosine[m]) + 2 * v2[m] * yhyhkconj.re[m])
        _gy9[m] = -u[m] * itapv1v2[m] * (-bhksine[m] + ghkcosine[m])
        _gy10[m] = -u[m] * itapv1v2[m] * (bhksine[m] - ghkcosine[m])
        _gy11[m] =
            u[m] *
            (-itapv2[m] * (-bhkcosine[m] - ghksine[m]) + 2 * v1[m] * itap2_yhyhkconj.im[m])
        _gy12[m] = -u[m] * itapv1[m] * (-bhkcosine[m] - ghksine[m])
        _gy13[m] = u[m] * itapv1v2[m] * (bhksine[m] + ghkcosine[m])
        _gy14[m] = u[m] * itapv1v2[m] * (-bhksine[m] - ghkcosine[m])
        _gy15[m] = u[m] * itapv2[m] * (bhkcosine[m] - ghksine[m])
        _gy16[m] =
            u[m] * (itapv1[m] * (bhkcosine[m] - ghksine[m]) + 2 * v2[m] * yhyhkconj.im[m])
    end
    nothing
end
```

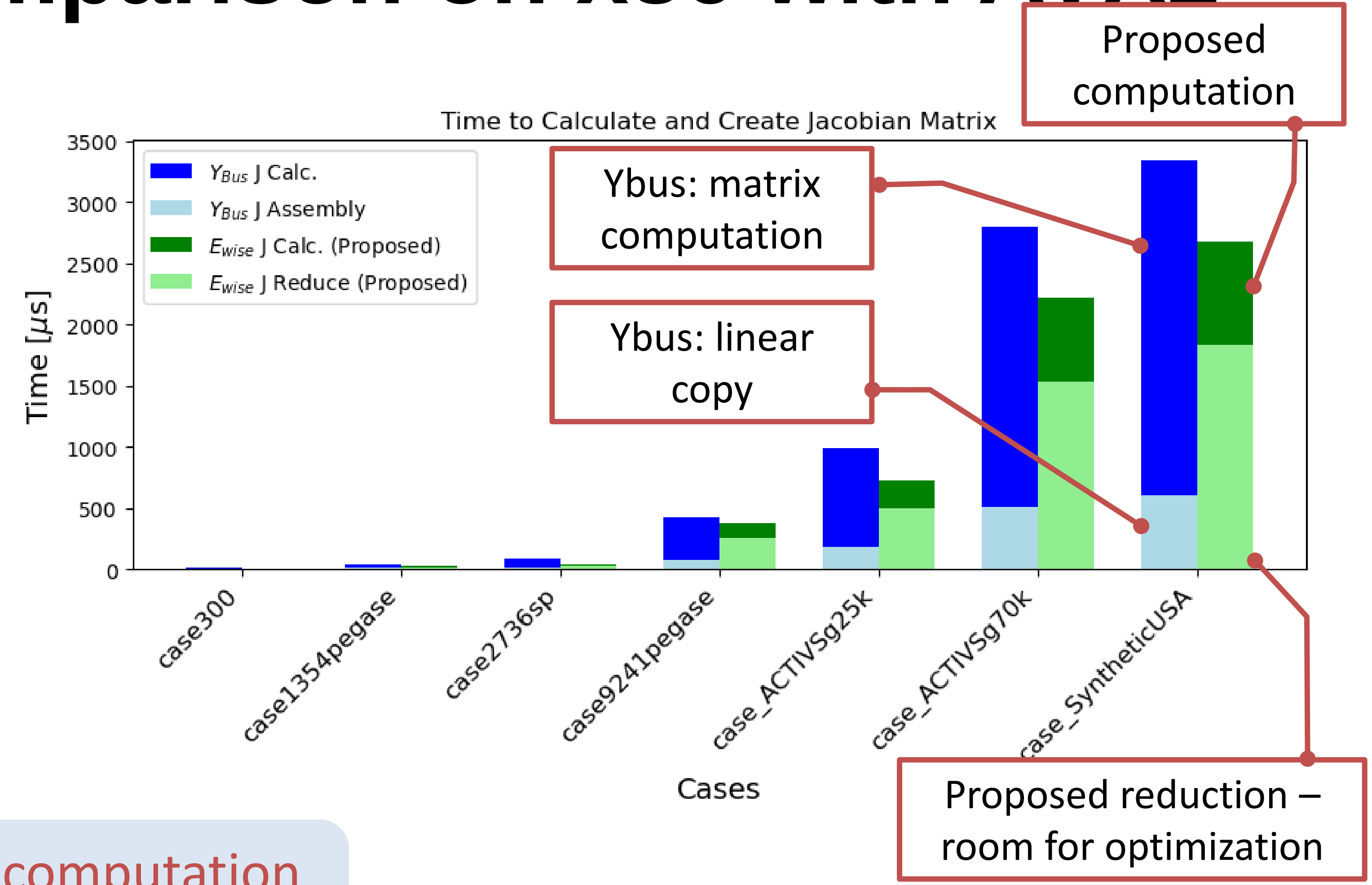
Single-instruction multiple data (SIMD)



Performance Comparison on x86 with AVX2

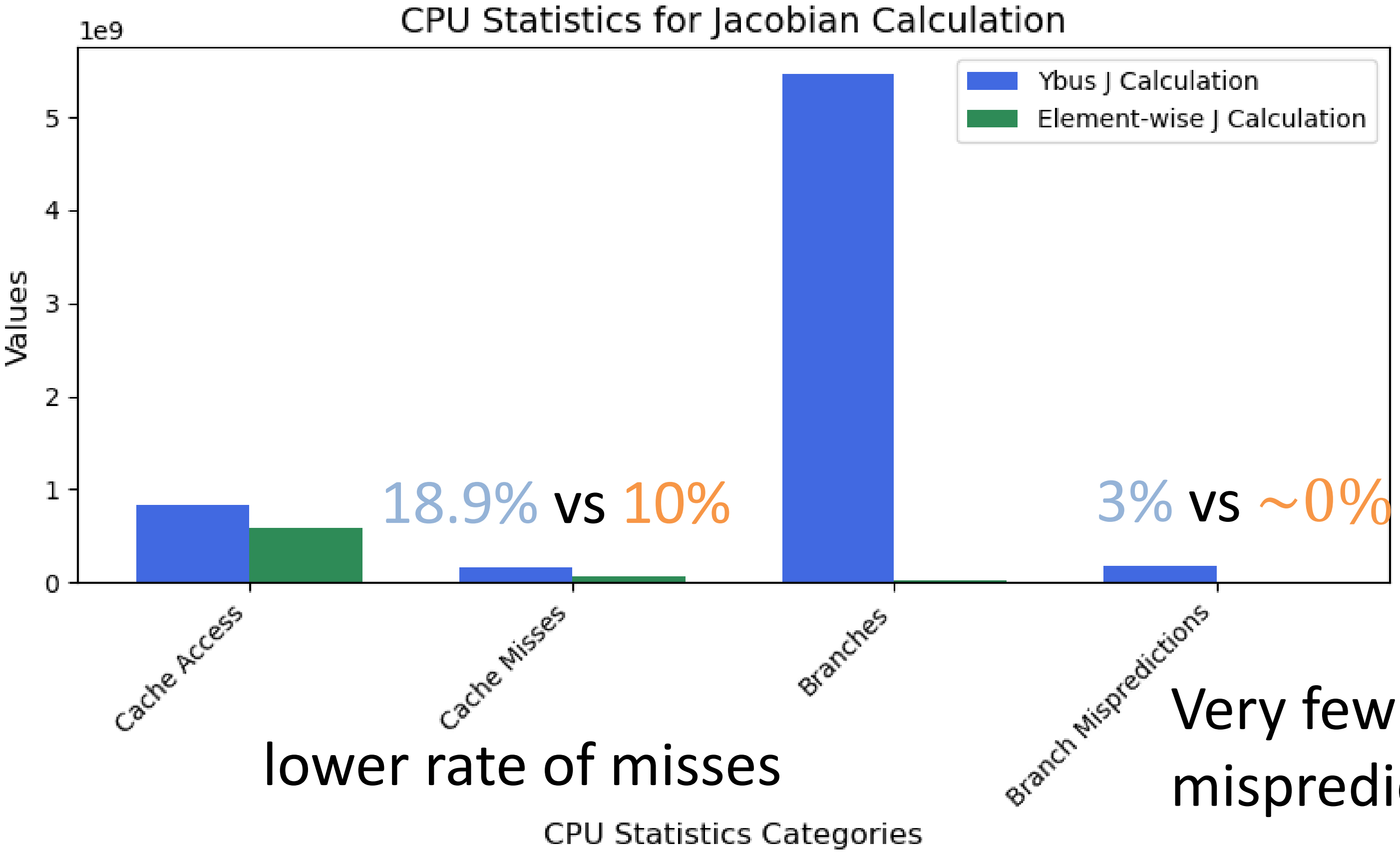
Total time for equation residuals & Jacobians





The median reduction in computation time is 30% with proposed method

Performance Data from `perf`



Very few misprediction

- Proposed element-wise method has computational advantage over Y_{bus} method
- Perspectives: Model reduction versus **Map + Reduce**
- Cultivating **multidisciplinary research** in power and computing

H. Cui, "Bus Admittance Matrix Revisited: Performance Challenges on Modern Computers," in *IEEE Open Access Journal of Power and Energy*, vol. 11, pp. 83-93, 2024, doi: 10.1109/OAJPE.2024.3366117.

Outline of this Talk

1. **A hybrid symbolic-numeric framework** for descriptive modeling and fast simulation
2. **An element-wise approach** for power flow calculation alternative to admittance matrix
3. **Ongoing studies** to unifying modeling and accelerating computation

Code Generation for Compiled Languages?

- Future power system simulation
  - Transient stability simulation + electromagnetic transient simulation
- Use ANDES as a modeling tool to generate optimized code
- Performance is the key
- The Julia case
  - Single Instruction Multiple Data (SIMD) vectorization on CPUs and GPUs
  - Multi-threading on CPUs
- Needs parallel-friendly data structure and computation workflow

H. Cui, F. Li and X. Fang, "Effective Parallelism for Equation and Jacobian Evaluation in Large-Scale Power Flow Calculation," in IEEE Transactions on Power Systems, vol. 36, no. 5, pp. 4872-4875, Sept. 2021, doi: 10.1109/TPWRS.2021.3073591.

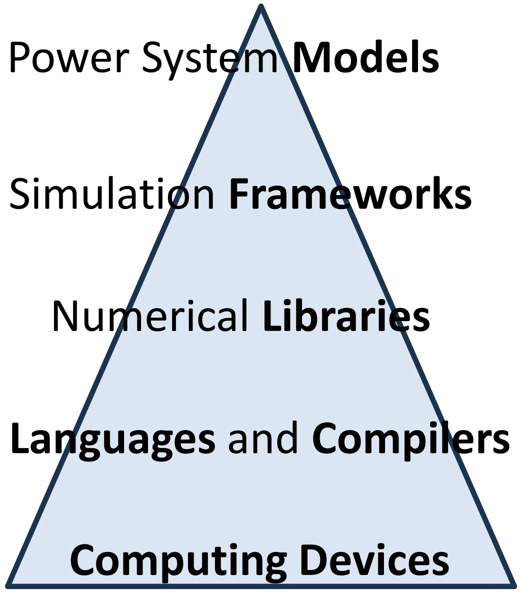
Ongoing Work 1: Transient Stability Simulation in the Julia Scientific Computing Ecosystems (1)

Motivation:

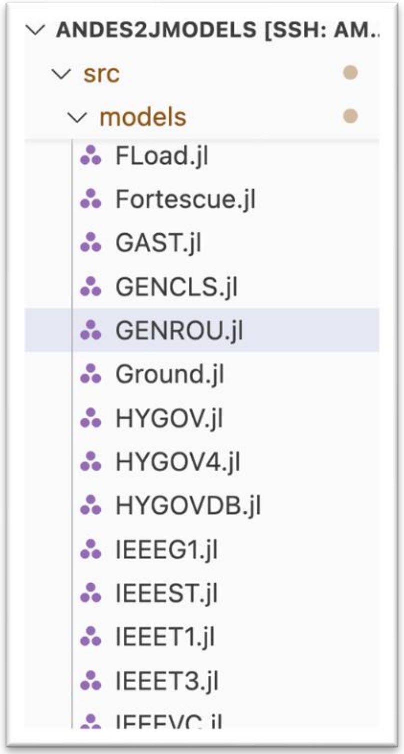
- Julia has the state-of-the-art DAE solvers
- Variable step based on error estimation; high-order & stiff-aware solvers
- To fully leverage hardware capabilities and sophisticated solvers and for large-scale stability simulation

State of the art:

- PowerSimulationDynamics.jl package (NREL)
- Current injection model, partitioned solution of DE and AE (may interfere with error estimation)
- Data structure does not support data parallelism



Ongoing Work 1: Transient Stability Simulation in the Julia Scientific Computing Ecosystems (2)

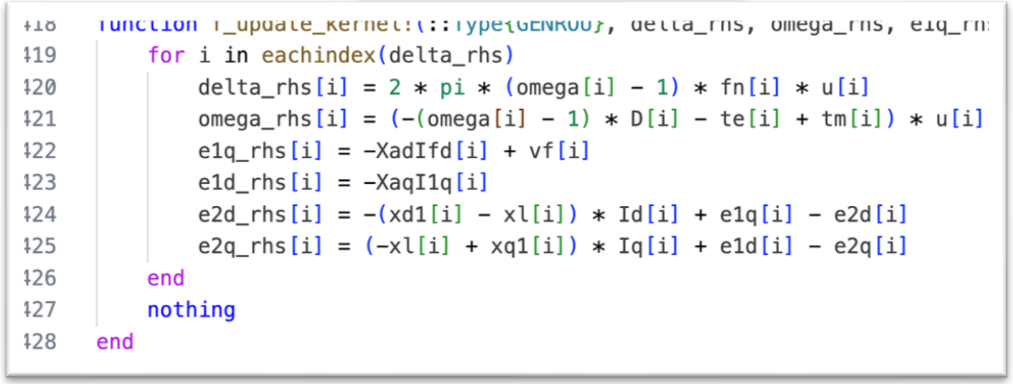


Research Question:

- Prevalent methods are fixed-step & low-order
- Is it more accurate and efficient to use high-order methods with variable step, error estimation, and compute interpolation?
- (Large test systems + complex methods)

A Systemic Approach:

- Generate parallel model implementations in Julia
- Write a framework to assemble models and interface with the solver

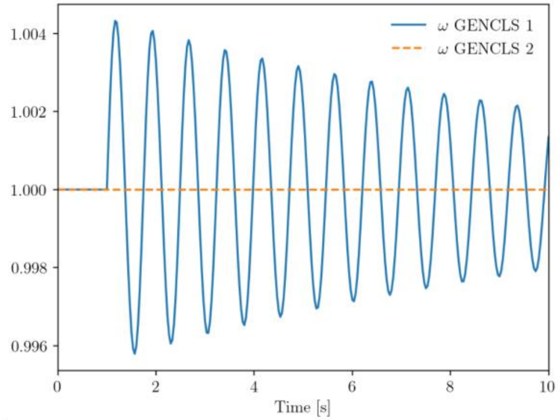


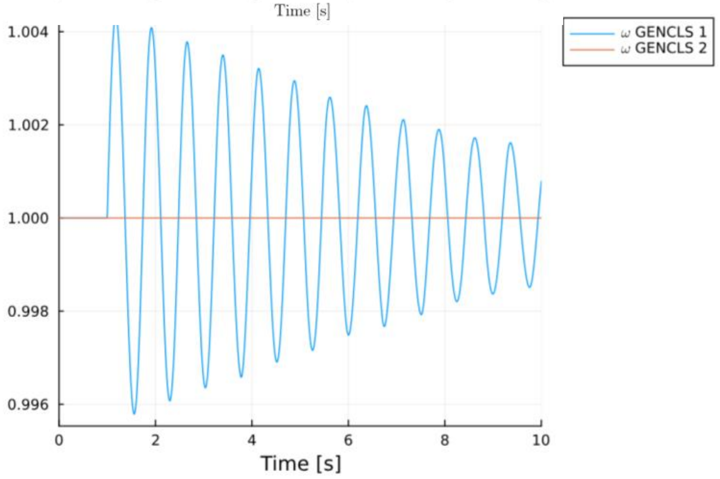
```julia
function f_update_kernel!(::Type{GENROU}, delta_rhs, omega_rhs, e1q_rh
    for i in eachindex(delta_rhs)
        delta_rhs[i] = 2 * pi * (omega[i] - 1) * fn[i] * u[i]
        omega_rhs[i] = (-(omega[i] - 1) * D[i] - te[i] + tm[i]) * u[i]
        e1q_rhs[i] = -XadIfd[i] + vf[i]
        e1d_rhs[i] = -XaqI1q[i]
        e2d_rhs[i] = -(xd1[i] - xl[i]) * Id[i] + e1q[i] - e2d[i]
        e2q_rhs[i] = (-xl[i] + xq1[i]) * Iq[i] + e1d[i] - e2q[i]
    end
    nothing
end
```

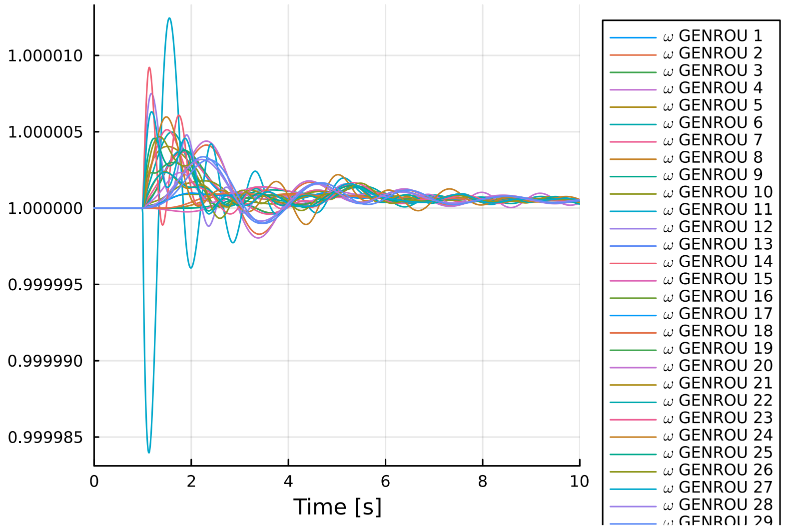
Ongoing Work 1: Transient Stability Simulation in the Julia Scientific Computing Ecosystems: Current Progress

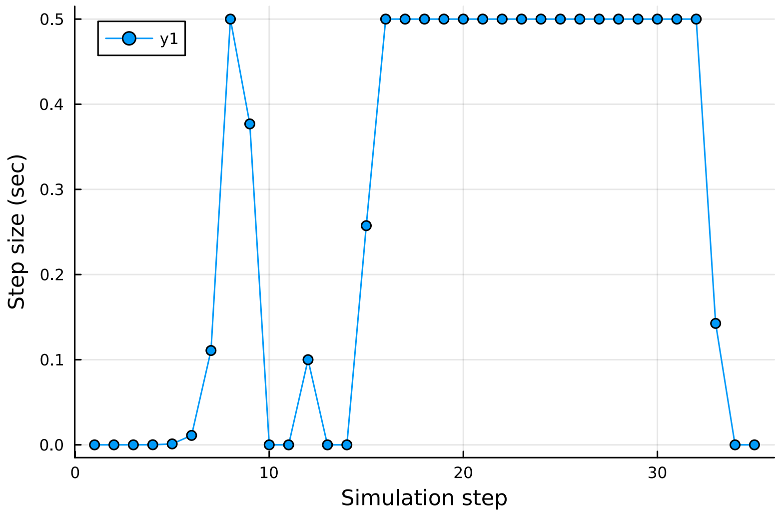
- Developed a *transpiler* to convert all model equations
- Develop mechanisms to support automatic differentiation for all models
- Programmed the "gluing" framework
- Leverages ANDES for data input and steady-state initialization

Results verified with ANDES using Single-Machine, Infinite-Bus system









181-bus, 29-machine system

69 ms per scenario!

```
t=1.0, Line 50 status changed to 0
t=10.0, Line connectivity statuses reset
  0.069180 seconds (408.80 k allocations: 29.749 MiB)
```

Ongoing Work 2: Multi-Timescale Simulation using Dynamic Phasor (1)

Background:

- Converter integration necessitates electromagnetic transient (EMT) simulation
- North American Electric Reliability Council issues new guidelines on EMT modeling

Objective:

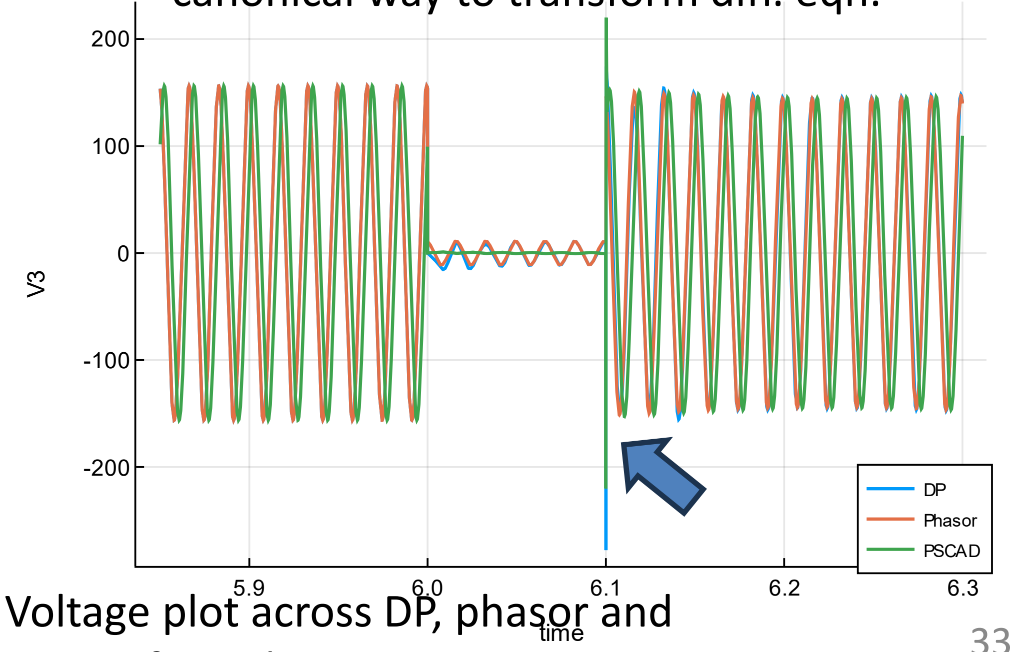
- Electromechanical and electromagnetic transients in one framework

Approach:

- Same solver infrastructure: variable-step + error control
- Dynamic phasor (DP) modeling – shift frequency to enable large step sizes

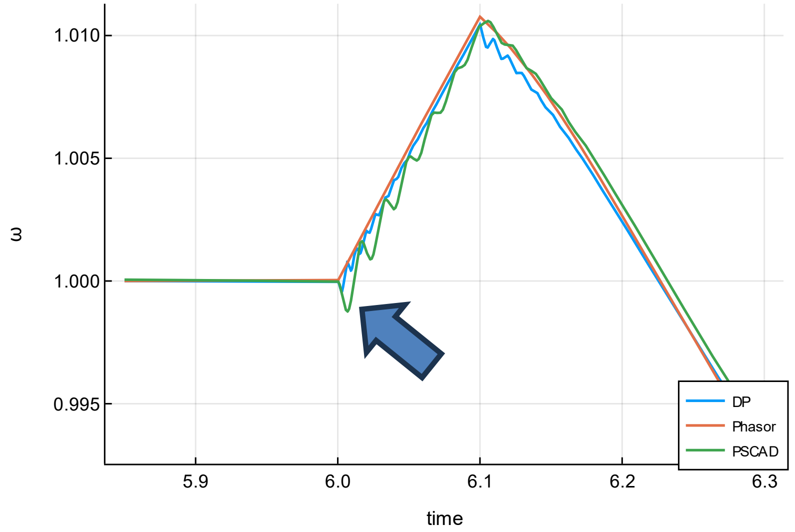
Mathematical Foundation

- For signal $x(t) = \tilde{x}(t)e^{j\omega t}$, $\tilde{x}(t)$ is known as dynamic phasor
- Derivative property: $\dot{x}(t) = \dot{\tilde{x}}(t)e^{j\omega t} + j\omega\tilde{x}(t)e^{j\omega t}$
- $\dot{\tilde{x}}(t) = \dot{x}(t)e^{-j\omega t} - j\omega\tilde{x}(t)$ -- there exists a canonical way to transform diff. eqn.

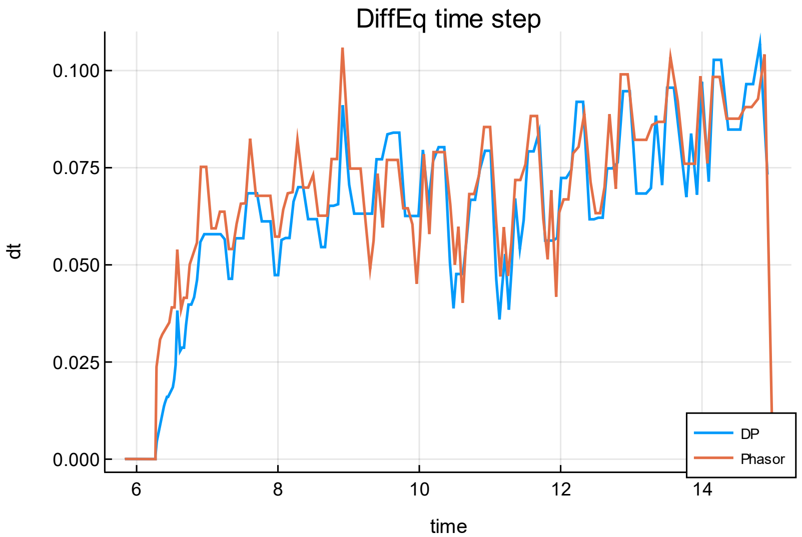


Voltage plot across DP, phasor and PSCAD formulations

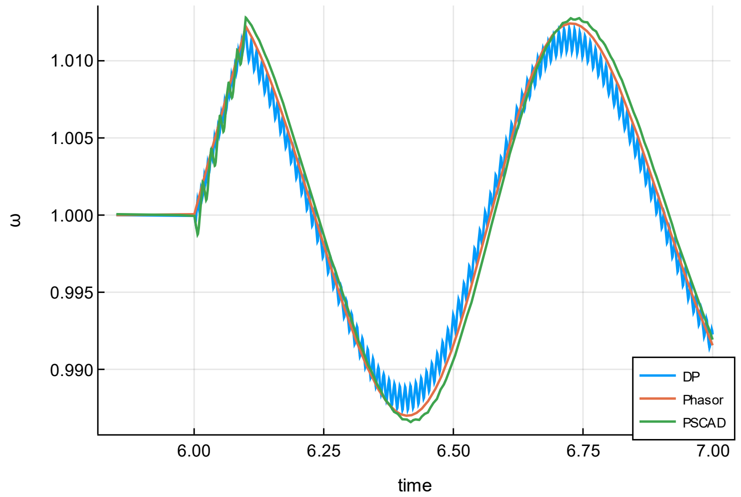
Ongoing Work 2: Multi-Timescale Simulation using Dynamic Phasor (2)



Generator speed comparison; both DP and PSCAD models capture the initial dip due to electromagnetic transients



Comparison of simulation step sizes in *sec*; DP step sizes are in the same order of magnitude as traditional phasor; PSCAD uses 500 μs fixed.



There is a catch! DP models are more oscillatory when system is *marginally stable* due to eigenvalue shifts along the Y axis. Still under investigation.

Summary

- ANDES introduces a **hybrid framework** that combines symbolic and numeric methods, enabling flexible, descriptive modeling for efficient power grid simulations.
- We discussed a **fine-grained parallelization** to accelerate computation of large-scale systems through optimized data handling and element-wise calculations.
- Leveraging Julia's scientific computing ecosystem, ongoing work focuses on high-order, variable-step methods for multi-timescale simulations.

Thank you!

Hantao Cui
hcui9@ncsu.edu