Goal

- Overall Goal: Determine how to stabilize the system before it collapses by running simulations of initial causes faster than real time by implementing the Parareal Algorithm

- Personal Goal: Parallelize the MATLAB code and determine speed up
Background

- **Power System:** Power lines with transformers, buses, generators, loads, etc.
- **Interconnected Systems:** East, West, Texas
- **High Performance Computing:** becomes necessary
- **Power Failures:** Creation and avoidance
Steady State System Simulation

- Determine voltage necessary to keep system at equilibrium
- Load amounts given, flat start (zero generation)
- Admittance Matrix

\[ Y = \begin{bmatrix}
  y_{11} + y_{13} & -y_{12} & -y_{13} \& 0 & -y_{13} \& 0 \\
  y_{12} + y_{23} & y_{12} & 0 & -y_{23} & 0 & 0 \\
  -y_{13} & -y_{23} & 0 & y_{13} + y_{23} + y_{34} & y_{34} & 0 \\
  
\end{bmatrix} \]
Steady State Solution

- Load buses (PQ), Generator Buses (PV), Slack Bus
- MatPower Solver, Newton’s Method
- Real and Imaginary Power:

\[ P_{i^{\text{sp}}} = P_i(\theta, V) = V_i \sum_{k=1}^{n} V_k (G_{ik} \sin \theta_{ik} + B_{ik} \cos \theta_{ik}) \]

\[ Q_{i^{\text{sp}}} = Q_i(\theta, V) = V_i \sum_{k=1}^{n} V_k (G_{ik} \cos \theta_{ik} + B_{ik} \sin \theta_{ik}) \]

- Solve for voltages and voltage angles
3 Generator 9 Bus Diagram
Dynamic System Simulation

- Initial Conditions: Steady State Values
- Create a fault: Downed line, generator outage, etc.

- Solve Differential and Algebraic Equations
<table>
<thead>
<tr>
<th>Name</th>
<th>Equation</th>
<th>MATLAB Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stator Algebraic Equation</td>
<td>[i \downarrow q @ i \downarrow d = 1/(R \downarrow a + X \downarrow d X \downarrow q)] [[R \downarrow a &amp; X \downarrow d @ - X \downarrow q &amp; R \downarrow a] [[E \downarrow q - V \downarrow q @ E \downarrow d - V \downarrow q]</td>
<td>Eq_SotorAlgebraic22.m</td>
</tr>
<tr>
<td>Network Algebraic Equations</td>
<td>[I \uparrow DQ = Y \uparrow DQ V \uparrow DQ] [Y \downarrow ij \uparrow DQ = [[\begin{bmatrix} B \downarrow ij + G \downarrow ij @ G \downarrow ij &amp; - B \downarrow ij \end{bmatrix};] [V \downarrow ij \uparrow DQ = [V \downarrow Qj / V \downarrow Dj ]; I \downarrow i \uparrow DQ = [I \downarrow Di / I \downarrow Qi ];]</td>
<td>NWAlgebraic22.m</td>
</tr>
<tr>
<td>Governor Model</td>
<td>[dP \downarrow SV / dt = 1/T \downarrow SV [- P \downarrow SV + P \downarrow C - 1/R \downarrow D S \downarrow m]]</td>
<td>Eq_SteamGov.m</td>
</tr>
<tr>
<td>Turbine Model</td>
<td>[dT \downarrow m / dt = 1/T \downarrow CH [- T \downarrow m + P \downarrow SV]]</td>
<td>Eq_SteamTurb.m</td>
</tr>
<tr>
<td>Change in q-axis Transient Voltage</td>
<td>[dE \downarrow q / dt = 1/T' \downarrow do [- E' \downarrow q + (X \downarrow d - X' \downarrow d) I \downarrow d + E \downarrow fd]]</td>
<td>Eq_ExcType1.m</td>
</tr>
<tr>
<td>Change in d-axis Transient Voltage</td>
<td>[dE \downarrow d / dt = 1/T' \downarrow do [- E' \downarrow d - (X \downarrow q - X' \downarrow q) I \downarrow q]]</td>
<td>Eq_ExcType1.m</td>
</tr>
<tr>
<td>Change in Exciter Field Voltage</td>
<td>[dE \downarrow fd / dt = 1/T \downarrow E [- (K \downarrow E + A \downarrow E (e \uparrow (B \downarrow E E \downarrow fd)) \uparrow) E \downarrow fd + V \downarrow R]]</td>
<td>Eq_ExcType1.m</td>
</tr>
<tr>
<td>Change in Rotor Angle</td>
<td>[d\delta / dt = w \downarrow B S \downarrow m]</td>
<td>Eq_Gen22.m</td>
</tr>
<tr>
<td>Change in Slip</td>
<td>[dS \downarrow m / dt = 1/2H [- D \downarrow S \downarrow m + T \downarrow m - T \downarrow e]] [dE \downarrow dc / dt = 1/T \downarrow e [- F \downarrow dc - (X' \downarrow q - X' \downarrow d) I \downarrow d]]</td>
<td>Eq_Gen22.m</td>
</tr>
</tbody>
</table>
Parareal in Time Algorithm

- Divides the time domain into intervals, and integrates concurrently over each interval.

- Used rather than spatial decomposition methods

- Coarse solve then fine solve in parallel
Methodology

- Coarse solution – Trapezoidal Rule (RK2)
- Fine solution - Runge-Kutta 4 method
  - Solve differential equations and algebraic equations in a predictor-corrector approach
- k1 – k4 equations
Pseudocode

Trapezoid Function Call – Initial coarse evaluation
While iterations less than max number of iterations:
  For each coarse section (in parallel):
    Runge-Kutta 4 Function Calls - fine evaluation
  Correct coarse evaluation
  Add one to iteration count

MATLAB ‘parfor’ tests

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Number of Loops</th>
<th>Serial Time</th>
<th>Parallel Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>32</td>
<td>84.986s</td>
<td>119.811s</td>
</tr>
<tr>
<td>10,000</td>
<td>64</td>
<td>267.242s</td>
<td>232.232s</td>
</tr>
<tr>
<td>5,000</td>
<td>1024</td>
<td>401s</td>
<td>314s</td>
</tr>
<tr>
<td>5,000</td>
<td>512</td>
<td>236s</td>
<td>262s</td>
</tr>
</tbody>
</table>
Results

- **Theoretical Speed up with 32 workers and 6 iterations:**
  
  \[
  \frac{32}{6} \sim 5.33
  \]

  - Where \( N \) is the number of parallel iterations running, \( k \) is the number of iterations to converge and \( N/k \) is the speed up

- **Measured Speed up with 32 workers:**
  
  \[
  \frac{5.423s}{1.151s} \sim 4.7
  \]

  - Where the numerator is the time for the serial loop to execute all iterations and the denominator is the time for the parallel loop to execute all iterations

- MATLAB parallelization overhead costs

- Conceptual Success
Conclusion/Future Work

- Steady state to dynamic system
- Parareal Algorithm implementation
- Personal goal accomplished: Parallelized MATLAB code and conceptually proved the speed up
- Optimize the program
- Parallelize the fault cases
- Benchmarking
- Create C/C++ Version
Sources


Questions?