Speeding up Simulation

Phil Rottier
Speeding up Simulation

- As models become increasingly complex, or the number of scenarios to be analysed increases:
  - “I have a big model and need to run long simulations as quickly as possible.”
  - “I need to perform design optimization as quickly as possible: which requires running many simulations as my design parameters change.”
  - “I need to perform thousands of Monte Carlo simulations as quickly as possible”
  - “My model takes forever to load”
Options for speeding up simulation

- Parallel and Distributed Computing
- Configuration settings
- Accelerator modes
- Model structure and content
- Smarter use of simulation
Options for speeding up simulation

- Parallel and Distributed Computing
- Configuration settings
- Accelerator modes
- Model structure and content
- **Smarter use of simulation**
Smarter use of simulation: what really matters?

- End to end project execution needs to fit within planned timescales
  - Understand the requirement
  - Understand the timescales
    - for development, test, V&V and documentation as well as execution

- Need to consider wider issues around people, process, tools
  - Understand the overall capability
  - Apply codes of practise

- If your simulation requirement is not a one-off
  - Understand and exploit opportunities for **component reuse**
  - Document, and learn from experience

- Execution speed of any given model is just one of many contributing factors
  - But it usually helps!
Example: get the requirement right

- Modelling an electrical power generation / distribution / propulsion system

- Taken from the requirement spec
  - ‘The model shall be capable of replicating the different configurations of the electrical plant … in steady state, transient and fault conditions.’
Example: get the requirement right

- **Alternative implementations**
  - Fully-populated with all switching functionality
  - Optimized with scenario-specific functionality
Example: get the requirement right

- Alternative implementation
Example: get the requirement right – how ??

- Adopt a structured approach
  - User requirements
  - System requirements
  - Design
  - Implement
  - Test and verify
  - Accept

- Adopt and apply **standards**

- Adopt and apply an agreed taxonomy
  - Version, variant, fidelity, accuracy, etc
Example: get the requirement right – how ??

- Work backwards from your objectives
  - What is the purpose ?
  - What scenarios need to be simulated to achieve the purpose ?
  - What measurements need to be made to achieve the purpose ?
  - With what fidelity will those measurements need to be made ?

- Hence
  - List of systems whose behaviour needs to be simulated
  - Appropriate level of fidelity with which those systems need to be modelled
  - Configurations and scenarios
  - Measurements and signals
  - Reports
Options for speeding up simulation

- Parallel and Distributed Computing
- Configuration settings
- Accelerator modes
- Model structure and content
- Smarter use of simulation
Model structure and content

- Use **requirements management interface** to help manage the linkage between the requirement set and the model(s)
- Facilitate development and team-working with **configuration control tools**
- Use **component libraries** to maximise opportunity for re-use, teamworking
- If model breaks up naturally into high-level sub-divisions, consider using **model reference** to partition the problem, spread the work amongst a team, enable parallel development, etc
- Consider use of **Simulink Projects** to manage the process
- Some modelling standards / best-practise can be codified and packaged in the form of **model advisor checks**
- “Bake in” the measurement of execution speed as part of the model development and test process
Example: Simulink Projects
Model reduction

- Consider use of other tools to minimise simulation complexity: design of experiments, statistical methods, optimisation methods
  - Models from test data with System Identification Toolbox
  - Statistical models with Model-Based Calibration Toolbox
  - Linear models with Simulink Control Design
  - Lookup tables to replace complex algorithms
  - Do the maths to implement a “reduced” variant of a modelling component

- If different levels of fidelity required for different applications of the model, consider implementing different fidelity variants, to minimise computing demand
  - Using variant subsystems
    - E.g. “average value” vs high-fidelity components
Example: average-value power electronic converter
Example: average-value power electronic converter
Model reduction: execution time comparison

- Full model
  - `>> tic; sim(bdroot); toc`
  - Elapsed time is 18.964078 seconds.

- Reduced model
  - `>> tic; sim(bdroot); toc`
  - Elapsed time is 4.415122 seconds.
Model reduction: results comparison
Model reduction: results comparison
Example: in-model measurements
Model structure and content

- If precise sim-time is not absolutely defined, consider using **conditional stop** to stop a simulation as soon as a desired condition is reached.

- If “running up” a model to a desired state is problematic, consider using **SimState** to save operating points for use as a starting point for subsequent runs.

- Build (semi-) automated test and verification into the development process:
  - E.g. exploiting MATLAB’s built-in **unit test framework** to manage test execution.
  - Time execution speed as part of the test framework.

- Consider use of **coverage / test generation** tools to assess completeness of testing.
Options for speeding up simulation

- Parallel and Distributed Computing
- Configuration settings
- **Accelerator modes**
- Model structure and content
- Smarter use of simulation
Accelerator modes

- Accelerator mode and Rapid Accelerator mode compile the model before execution
- Compiled code speeds up simulation of many models
- Works best with models where run time is long compared to the compilation time
- Integrated seamlessly into the tools
  - Easy to try
Accelerator mode

- Converts a model into a binary DLL, removing the block-to-block overhead of a normal block-by-block interpreted simulation
  - Less time is typically spent in the Simulation stage and more time is typically spent in the Initialization stage of the simulation

- Considerations
  - If the model primarily contains blocks which are compiled S-functions, there may not be any significant speedup
  - Run time diagnostics disabled
Rapid Accelerator mode

- Converts a solver and model into an executable, running as a separate process
  - If possible, runs on a separate core from the MATLAB session
  - Leveraging two cores may outweigh the overhead of inter-process communication

- Considerations
  - The entire model must be capable of generating code
  - S-functions not supported
  - Run time diagnostics are disabled
  - Signal logging is disabled
Accelerator modes - example

```matlab
%Simulation modes
modes = {'Normal', 'Accelerator', 'Rapid-accelerator'};

%For each mode in turn
for i = 1:numel(modes)
    %Set mode
    set_param(mdl, 'SimulationMode', modes(i));
    %Run it, and time it (twice, so second time excludes build time)
    for j = 1:2
        %Start timer
tic;
        %Run the model
        sim(mdl);
        %How long did it take?
        T(i,j) = toc;
    end
end
```
Accelerator modes

- Supported features

<table>
<thead>
<tr>
<th>Capability</th>
<th>Mode:</th>
<th>Normal</th>
<th>Accelerator</th>
<th>Rapid Accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tunable parameters</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Signal Logging</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support s-functions</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supports interpreted MATLAB code</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handles algebraic Loops</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debugging</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Profiling / Perf. Advisor</td>
<td>✓</td>
<td></td>
<td>(✓)</td>
<td></td>
</tr>
<tr>
<td>Supports SimState</td>
<td>✓</td>
<td></td>
<td>(✓)</td>
<td></td>
</tr>
<tr>
<td>Model Coverage</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime Diagnostics</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Accelerator modes

- Considerations
  - Model may contain blocks, such as Interpreted MATLAB, that cannot be compiled
  - Model may already contain compiled code, e.g. s-functions, that cannot be sped up further
  - End to end execution speed may be due to other phases than the Simulation phase
  - End to end execution speed may be due to memory or numerical issues
Options for speeding up simulation

- Parallel and Distributed Computing
- **Configuration settings**
  - Accelerator modes
  - Model structure and content
  - Smarter use of simulation
Configuration settings

- Consider
  - Using Inline Parameters
  - Implement logic signals as Boolean
  - Conditional branch execution – enabled and triggered subsystems

- Diagnostics with substantial overheads:
  - Solver data inconsistency (for S-functions)
  - Stateflow & MATLAB debugging & animations
  - Data range / overflow diagnostics

- Solver selection
Enable Inline Parameters Optimization

Why would Simulink speed up?

- Enabling the Inline parameters optimization lets Simulink use the numerical values of model parameters
  - Thus Simulink does not need to support tuning them at run-time
- Other optimizations can leverage the fact that the values are known and can reduce the required mathematics

What’s the tradeoff?

- When enabled, user must explicitly specify variables that will be used to tune block dialog parameters during simulation, hence these variables need to be identified before start of simulation

Help Search: inline parameters
Disable MATLAB Debugging

Why would Simulink speed up?
- Disabling debugging, overflow detection, and echoing to the command line

What’s the tradeoff?
- This makes it difficult to debug issues that dynamically arise
Avoid Simulation Animations

Why would Simulink speed up?
- Closing Animations decrease CPU load
- Open charts, including those that are minimized, slow the simulation
- Animations can also be disabled

What’s the tradeoff?
- Animations can be very useful for debugging and gaining insight
- Disabling Stateflow debugging/animation also disables MATLAB debugging

Help Search: disable Stateflow animation
Ensure that Expensive Debugging Diagnostics are disabled

**Why would Simulink speed up?**

- The following diagnostics can noticeably slow down the simulation speed:
  - Solver data inconsistency
  - Array bounds exceeded
  - Some data validity diagnostics
- By default, these diagnostics are disabled

**What’s the tradeoff?**

- It may be desirable to enable the above diagnostics, especially when debugging custom S-functions
- The effect of disabling other diagnostics on simulation time is typically negligible thus disabling other diagnostics is not justified

*Help Search: solver diagnostics*
Enable C Compiler Optimizations

Why would Simulink speed up?
- Configuring the C compiler to use optimizations (option off by default) when compiling code typically resulting in faster simulations

What’s the tradeoff?
- Code compilation typically takes longer when optimizations are enabled
- Only external compilers will be affected by this option (i.e., this option has no effect if using LCC)
Solver selection

- Continuous or Discrete?
- Variable or Fixed?
- Stiff or Non-Stiff?
- Order of Accuracy / step size limitations
- Zero-crossing setting?

### Variable-Step Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>Problem Type</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode45</td>
<td>Nonstiff</td>
<td>Medium</td>
</tr>
<tr>
<td>ode23</td>
<td>Nonstiff</td>
<td>Low</td>
</tr>
<tr>
<td>ode113</td>
<td>Nonstiff</td>
<td>Low to high</td>
</tr>
<tr>
<td>ode15s</td>
<td>Stiff</td>
<td>Low to medium</td>
</tr>
<tr>
<td>ode23s</td>
<td>Stiff</td>
<td>Low</td>
</tr>
<tr>
<td>ode23t</td>
<td>Moderately Stiff</td>
<td>Low</td>
</tr>
<tr>
<td>ode23tb</td>
<td>Stiff</td>
<td>Low</td>
</tr>
</tbody>
</table>

### Fixed-Step Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>Integration Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode1</td>
<td>Euler</td>
</tr>
<tr>
<td>ode2</td>
<td>Heun</td>
</tr>
<tr>
<td>ode3</td>
<td>Bogacki-Shampine</td>
</tr>
<tr>
<td>ode4</td>
<td>Runge-Kutta</td>
</tr>
<tr>
<td>ode5</td>
<td>Dormand-Prince</td>
</tr>
</tbody>
</table>

### Implicit-Step Solver

ode14x
Solver selection

- **Fixed step**
  - User-specified step-size, determines simulation time and numerical accuracy

- **Variable step**
  - User-specified tolerance, Simulink determines step-size
  - Opportunity for significant speed-up without loss of accuracy

- **Stiff or Non-Stiff?**
  - Stiff solver required for systems with wide-ranging dynamics

- **Many different solver options**
  - Doc Search: “Choosing a Solver”
Solver selection: example

- For SimPowerSystems users
  - Type “power_new” to create a blank modelling canvas
  - Including a utility to help optimise settings
Measure the effect: don’t just apply and hope for the best

- Use `get_param` / `set_param` to manipulate the model programmatically
- Create a benchmarking script
- Run it regularly
  - E.g. every time you commit changes to revision control
Measure the effect: example

tic;
sim(mdl);
T = toc;

%Do it again (to measure difference between build and re-build time)
tic;
sim(mdl);
T(end+1) = toc;

%Apply changes to settings
set_param(mdl, ...
    'SimulationMode', 'Accelerator', ...
    'InlineParams', 'on', ...
    'SFsimEnableDebug', 'off');

%Run it again
tic;
sim(mdl);
T(end+1) = toc;

%Do it again (to measure difference between build and re-build time)
tic;
sim(mdl);
T(end+1) = toc;

%Report
barh(T);
set(gca, 'ydir', 'reverse', 'ytick', 1:numel(T), 'yticklabel', ...
    {'Build and run', ...
     'Rebuild and run', ...
     'Build and run (optimised)', ...
     'Rebuild and run (optimised)'})
Simulink Profiler
Identify bottlenecks in all phases

- The Simulink Profiler will tell you how much time is spent in each phase of simulation
- Find out which blocks take the longest to simulate
Debugger

- Step through model block by block
- Set breakpoints
- Interrogate blocks for state, output, etc
Performance advisor

- Helps to automate the workflow
  - Create a baseline point of reference
  - Checks models for conditions and settings that result in inaccurate or inefficient simulation or code generation
  - Apply changes to improve efficiency
  - Re-run and compare
- Contains performance and accuracy checks
- Report can be process artifact
Options for speeding up simulation

- Parallel and Distributed Computing
  - Configuration settings
  - Accelerator modes
  - Model structure and content
  - Smarter use of simulation
Parallel and Distributed Computing

What you CAN do with Simulink and Parallel Computing

- Reduce the total amount of time it takes to
  - Run multiple independent simulations
  - Generate code for multiple referenced models
- You can do these tasks on a
  - Local multi-core computer with Parallel Computing Toolbox (PCT)
  - Computer cluster with MATLAB Distributed Computing Server (MDCS)

What you CANNOT do with Simulink and PCT

- Split one model into multiple sections and run that model in parallel
- Build distributed code for a multi-core target
Parallel computing options

Options to run Simulink models in parallel
- MATLAB scripts that use `parfor` and `parfeval`
- Simulink Design Optimization with parallel computing option enabled
- System Test with parallel computing option enabled

Other products that work with parallel computing tools
- Simulink Coder (build code in parallel)
- Model-Based Calibration Toolbox (build models in parallel)
MATLAB Distributed Computing Server

- Run Simulink® models on computer clusters, clouds, and grids.
- Develop your model on a multicore computer using Parallel Computing Toolbox and then scale up to many computers by running it on MATLAB Distributed Computing Server.
- The server supports batch jobs, parallel computations, and distributed large data.
- The server includes a built-in cluster job scheduler and provides support for commonly used third-party schedulers.
- MATLAB Distributed Computing Server provides licenses for all MathWorks® toolboxes and blocksets
  - So you can run your MATLAB programs on a cluster without having to separately acquire additional product-specific licenses for each computer in the cluster.

MATLAB Digest Article: Improving Simulink Design Optimization Performance using Parallel Computing
Parallel and Distributed Computing – issues to consider

- **Same** data required by different workers
  - May be expensive to recreate on each – negating the benefit of parallel
  - See “Worker Object Wrapper” on MATLAB Central

- **Guaranteed different** data required by different workers
  - E.g. random seed for Monte Carlo studies
  - Pass required seed value(s) in to each worker, and ensure model handles them accordingly
Useful links

- Webinar for Speeding up Simulink: http://www.mathworks.com/wbnr49883
- Webinar with signal-processing focus: http://www.mathworks.com/wbnr50508
- Tech. support list of “thing to check”: http://www.mathworks.co.uk/support/solutions/en/data/1-18AAG/index.html
Summary: Options for speeding up simulation

- Parallel and Distributed Computing
- Configuration settings
- Accelerator modes
- Model structure and content
- Smarter use of simulation