Accelerated Simulations Using Real-Time Workshop and the Rapid Simulation Target

Abstract:

Real-Time Workshop's Rapid Simulation (RSim) Target, available in MATLAB Release 11, achieves dramatic speedup of Simulink fixed-step simulations. The RSim Target enables you to use generated code to create fast, compiled simulations that also read new signal data and parameter data directly from MATLAB MAT-files without needing to recompile your model. An RSim simulation lets you run accelerated simulations on either your host computer, or on other similar computers. Test results show up to an 18 times speed improvement over Simulink simulations using the RSim Target. When many simulation runs are needed, additional speedup is possible by using distributed computing with RSim executables running on multiple independent computers to achieve maximum simulation performance.

Fast, Portable Simulations

Real-Time Workshop generates C code for your Simulink block diagram and uses your C compiler to create an executable that is capable of increasing simulation performance by a factor of up to 18 times speed improvement for large models. Actual performance will vary. Factors that affect the overall time to run a simulation include the size of the model (the number of blocks as well as the number of operations required by the set of blocks in your model) and the number of time steps needed to complete the simulation.

In aerospace, automotive, communications, and other industries, users are often confronted with both large models and large numbers of time steps making simulation time particularly long. In such applications the advantages provided by an RSim simulation are most apparent.

The reason why RSim simulations created with Real-Time Workshop run faster than normal Simulink simulations is that the simulation is performed using compiled C code. The C code is highly optimized for your model. When you change your model structure, new C code is generated, compiled, and linked. The use of a compiled model-specific C code is a valuable initial step for improving speed. If your needs include running many simulations with the same model while using either different signal data or parameter data, your application could also benefit by running independent simulations on several computers. This combined approach using model-specific C code and distributed computing—each simulation running a separate data set—has the ability to provide speedup of several orders of magnitude over a standard Simulink simulation running on a single computer.

Below is a table providing some comparisons of speedup using one computer. Speedup was measured by running Simulink and RSim simulations from the MATLAB command line using the following M-file.

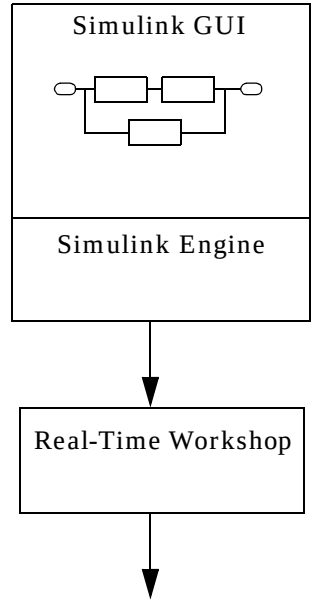
```
tic
sim('model')   % Run Simulink simulation
a = toc        % Get elapsed time for Simulink simulation
tic
!modelname     % Run an RSim simulation (modelname.exe)
b = toc        % Get elapsed time for RSim simulation
x = a/b        % Determine speedup ratio
```

Table 1: Speedup Factor Computed for Three Simulink Models

Model	Speedup using RSim
`dspafsf demo`	2X (very small DSP model with only 8 blocks and one CMEX S-function)
`f14 demo`	6X (small model with 39 blocks)
Spacecraft model	18X (large model with over 1,000 blocks)

Architecture

To understand why Rapid Simulation Target simulations are faster than normal Simulink simulations, which are considered interpretive simulations, let's explore the architecture of Simulink and Real-Time Workshop.



Compiled Model Specific C code

Simulink GUI enables you to build diagrams using your mouse.

Simulink Engine transforms block diagrams into a format that can be simulated. Once the transformation is complete, you can perform normal simulations using an interpretive scheme that lets you quickly start, stop, and restart simulations even after signfiicant model edits such as adding or deleting blocks.

Real-Time Workshop generates C code, and invokes your compiler to create an executable.

The Rapid Simulation Target is built upon Real-Time Workshop. Real-Time Workshop is a tool that transforms Simulink block diagrams into generated code for your selected target. In this case, the target is the Rapid SimulationTarget. Once code is generated, the code is compiled and linked using your C compiler. Rapid Simulation Target delivers significantly faster simulations.

In this paper we refer to the expression *compiled model specific C code* to refer to an executable created for the Rapid Simulation Target. This executable is created using Real-Time Workshop.

Another important term is *parameter transformation*. In Simulink, you can create a block diagram that uses numeric values, variables, or parameter transformations to correspond to the “value” or “values” to be used in a simulation. Parameter transformations can arise when you have any type of expression such as 2*3, 2*A, sin(2*pi*alpha). Each of these expressions is evaluated to a numeric value in the form of a scalar, vector, or matrix. In Simulink you use Ctrl-D to update a block diagram. During this process, parameter transformations are evaluated so that the corresponding value(s) can be used in a simulation. This information is useful in understanding differences between Simulink simulations and simulations using the Rapid Simulation Target.

The Real-Time Workshop Generic Real-Time Target

Before discussing RSim in detail, let us first review the Generic Real-Time (GRT) Target to better understand the distinction between the GRT Target and the RSim Target. Although GRT can be used to create a fast, stand-alone executable, the RSim Target is preferred for this type of application.

GRT stands for Generic Real-Time. The reason why we use the term “generic” is that the set of target files for GRT does not include any hardware-specific code for interrupt service routines (ISRs), operating systems, or other target-specific timing devices. Without any modifications GRT allows you to exercise generated code as a nonreal-time simulation, for example, simulating your model on your host computer and logging data to a MAT-file. This addresses the initial purpose for GRT. GRT lets you confirm that the generated code performs correctly and provides the same results as when running a fixed-step simulation in Simulink.

The second purpose for GRT Target files is for use as a starting point or set of templates that you can modify in order to create your own custom target to run in real time on your selected hardware. GRT Target files include comments that instruct you where to modify code in order to enable or disable interrupts when writing your own ISR to control real-time execution of the model code. Once you make these modifications to a set of target files (after renaming them), you now have a custom target rather than a generic target. Your custom target, based on the GRT Target, is suited for real-time execution in a rapid prototyping or hardware-in-the-loop application.

Since the intended use of the GRT Target is real-time execution, certain assumptions are made about the target. First of all, GRT assumes that a file system may not be available. Because of this assumption, signal data for all targets other than the RSim Target is placed within the generated code for both From File and From Workspace block data, thereby eliminating the need for functions to read data from a file. For most real-time systems, you would normally avoid the use of From File or From Workspace blocks and use an I/O device such as an analog to digital converter to measure signal data at each sample interval. You would also avoid the use of MAT-file data logging in real-time systems for the same reason. However, the GRT Target allows you to select MAT-file data logging so you can compare simulation results from generated code on your host computer against your Simulink simulation results.

GRT has another valuable capability. With the `grt_main.c` file, function calls are included that are ready to use with external mode. External mode allows you to change parameters on-the-fly which is particularly useful in rapid prototyping systems. With external mode, you open the dialog box for a particular block for which you would like change parameter values and simply type the new value. External mode does the rest. New parameter values are then exported to the external model via TCP/IP, or some other communication transport where they are installed while the simulation continues to run.

Additional targets supplied with Real-Time Workshop are intended for other special purposes. For example, the Embedded Real-Time (ERT) Target serves as a starting point when you want to use generated code in an embedded application requiring minimal memory usage. The ERT Target provides significantly tighter code generation requiring less RAM and ROM as necessary in an embedded application.

When using a model containing From File blocks, the GRT Target places From File data into the intermediate `model.rtw` file. Next, the Target Language Compiler (TLC) reads the `model.rtw` file during generation of the C code. This C code also includes From File block data.

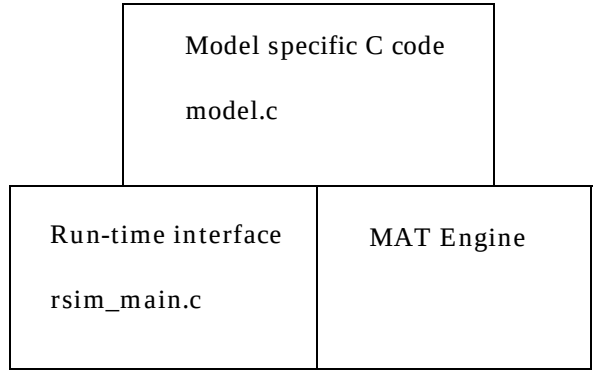
Running stand-alone simulations using the GRT Target and From File blocks also has a performance issue during code generation. Placing data into the generated C code takes additional time. Compilation is also slower since signal data is also passed through the compiler. This is particularly noticeable for models with large From File block data sets. Perhaps the most compelling reason not to use the GRT Target for stand-alone simulations is that signal data for From File blocks cannot be changed from one simulation to the next without recompiling.

In comparison, the RSim Target was created explicitly for the purpose of creating fast, stand-alone (nonreal-time) simulations that read new signal and parameter data for each simulation run. Data is read directly from MATLAB MAT-files for maximum convenience without the need to recompile your model between runs.

Using the Rapid Simulation Target

The Rapid Simulation (RSim) Target consists of a set of files that are included with Real-Time Workshop and ready for use without modification. These files result in subtle but important differences in the code that is generated from a Simulink model. The RSim Target's system target file `rsim.tlc` is used by the Target Language Compiler to place function calls in the generated code to perform MAT-file open, read signal and parameter data, check signal and parameter consistency, MAT-file close, and so on. Once you've selected the RSim Target, the generated code accommodates changes necessary for improved handling of From File, To File, and From Workspace blocks.

Template makefiles are included for Watcom C, Microsoft Visual C, Borland C, and UNIX C compilers and are ready to use without modification. You can manually select a template makefile or allow Real-Time Workshop to use a makefile for the C compiler you previously selected for building MEX-files. When code generation is completed, the RSim template makefile immediately invokes your C compiler to create an executable that you can run stand-alone.



An RSim simulation consists of models as shown.

At the start of the simulation, the file `rsim_main.c` looks for optional user-supplied arguments that specify replacement filenames containing new signal or parameter data. Then `rsim_main.c` runs the simulation by incrementing the simulation time and running the function `rt_onestep` that evaluates blocks at each sample interval. Data logging is also controlled from rsim_main.c and upon completing a simulation, logged data is stored to a MAT-file. As with other files provided with the RSim Target, there is no need to modify `rsim_main.c`, it is ready to use.

How Does RSim Handle Input Signal Data?

The RSim Target handles From File blocks quite differently than other Real-Time Workshop targets. RSim assumes you will run the generated code on a desktop computer or workstation where a file system is always available. This eliminates the need to embed From File block data into the generated code, which means code generation occurs much faster and compilation time is also reduced. With the RSim Target From File block data is read at run time from a MATLAB MAT-file during initialization.

The MAT-file containing signal data is virtually ignored during code generation with the exception that RSim checks and stores information about the width of input signals used in From File blocks. This ensures correct treatment in the generated code for blocks and signals that are subject to loop rolling, for example, handling vectorized block inputs and outputs. However, if your RSim simulation contains From File blocks, the MAT-file(s) must be present so that the data can be read at the start of the simulation. If your model does not contain any From File blocks, then a corresponding MAT-file with data for a From File block is not needed.

Running an RSim Simulation for Models Containing From File Blocks

Let's assume that your model includes one or more From File blocks. The RSim model opens the MAT-file(s), reads the data, and runs the simulation to completion. Assume you've already generated a RSim executable for your model named `mymodel` and you have MAT-file called `input.mat` containing From File block data. You can run your simulation from the system command prompt by typing the model name.

```
mymodel
```

Your simulation runs until reaching the stop time specified in the dialog box at the time the code was generated. Assuming you selected MAT-file logging options prior to generating code, simulation results are saved to a file named `mymodel.mat`.

You can also specify a replacement filename for the From File block MAT-file as a command line argument. To use `mynewinput.mat` as the replacement From

File block data, you would use the `-f` flag (`f` corresponding to “From File”) as follows.

```
mymodel -f input.mat=mynewinput.mat
```

In this case your simulation reads From File block data from the file `mynewinput.mat` and checks to insure that the data is compatible. For example, the new data must contain the same number of inputs signals as existed when generating the model code. However, it is not necessary to keep the signal length the same.

Data for From File blocks is read only at the initialization of the simulation. This makes RSim well suited for running batch simulations. You can gather sets of signal data, store data in MAT-files, and issue commands that repeat a series of simulation runs using new data sets for each simulation run.

Changing Parameter Data and From Workspace Data with RSim Simulations

Simulink includes a highly flexible feature calling masking. A mask lets you alter the appearance of a block and utilize MATLAB parameters and scripts that are executed upon initialization of the simulation. During code generation, MATLAB scripts and MATLAB workpace variables are evaluated and the resulting values are mapped into the `model.rtw` file. Ultimately these values are placed in the generated C code. MATLAB scripts cannot be re-evaluated by the generated code since MATLAB is necessary to parse and execute a MATLAB script or M-file.

Simulink provides powerful and extensive block parameter handling capabilities. For example, you can specify a Gain block parameter as a numeric value, MATLAB variable, or an arbitrary MATLAB expression. When the parameter is not a numeric value, MATLAB evaluates the parameter to determine a numeric value (or values when a matrix may result) and uses the numeric values during simulation. Operations that evalute parameter expressions are referred to as parameter transformations. Simulink masking is another feature that may introduce parameter transformations.

Parameter transformations must be evaluated using MATLAB and Simulink in order to obtain the correct mapping of a value to a memory location in the generated code. To achieve the correct mapping, RSim includes an M-file called `rsimgetrtp`. This function updates the block diagram (same as Ctrl-D), extracts the entire parameter structure, and allows you to save it to a MAT-file for later use in an RSim simulation. This parameter structure provides support

for all of Simulink's intrinsic data types including; `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `float`, and `double`. It also supports complex value data in any of these data types. You can change any parameter value including parameters used in masks located anywhere in the model. This can be done by manually changing the value or it can be done programatically using an M-file.

For a better understanding, let's consider a simple example where a dialog box uses a very basic parameter transformation *2*A*. Assume that the variable "*A*" is defined as a scalar in the MATLAB workspace. You can simply change the value of *A* in MATLAB (e.g., *A=5*) and then run the M-file `rsimgetrtp`. Simulink evaluates all parameters and masks throughout the block diagram and returns the `rtP` structure to the MATLAB workspace. The following illustrates how to use rsimgetrtp to save the parameter structure to a MAT-file `myparams.mat`.

```
rtP = rsimgetrtp('mymodel');
save myparams rtP;
```

The RSim executable can later read parameter MAT-files at the start of execution, and compare model checksums to ensure that the set of blocks and block connectivity matches at the time of code generation. If the checksums match, the new parameters are loaded and the simulation runs. You can run this by typing the following from MATLAB.

```
!mymodel -p myparams.mat
```

In the event that a parameter file was not provided, the model would execute using the orignal set of parameter values within the executable. The parameter MAT-file is entirely optional and only needs to be used when you want to change a parameter value in the RSim model.

Extending this example, you could create a script that changes model parameters, runs `rsimgetrtp` to extract a new `rtP` parameter structure from the Simulink block diagram, and stores the parameter structure to a MAT-file. This can be repeated within a loop using unique names such as `myparams_1.mat`, `myparams_2.mat`, and so on to create multiple sets of data for later use in RSim simulations. Real-Time Workshop includes a demo examples rsimdemo1 and rsimdemo2 which include M-file scripts that illustrate precisely how this is done. You may want to use these as a starting point that you edit for your own purposes.

You can run a set of RSim simulations from the MATLAB command line, from DOS, or from a UNIX prompt. Scripts, for example, `.bat` file, can be created to

automatically run a series of RSim simulation runs. And you can also write M-files that use bang (!) to run multiple simulation runs directly from MATLAB.

When using the From Workspace block, RSim treats signal data as parameters. This differs significantly from the From File block. Since the From File block requires signal data to be available in a separate MAT-file in addition to the executable, you must have the signal data file available to run a simulation. In contrast, the From Workspace block data is placed in the generated code as a parameter. This allows your RSim executable to run without any additional MAT-files. There are subtleties when using the From Workspace block with the RSim Target. Code generation and compilation time are slower since signal data is placed in the `model.rtw` file, in the generated code, and then compiled. Also, it is not possible to change the length of the signal data used for the From Workspace block in subsequent RSim simulations. But, with the From Workspace block, you do have the flexibility of having masked blocks or other blocks that have parameter transformations. This is not the case with the From File block.

In summary, the length of signals used in the From Workspace block directly impacts code generation and compilation time. Signal lengths for From Workspace blocks must remain constant when using RSim. Parameter transformations are handled by using `rsimgetrtp` to extract the new `rtP` parameter structure. Generation of the new rtP structure requires that you use MATLAB, Simulink, and Real-Time Workshop in order to run `rsimgetrtp`.

In contrast, From File block data does not support parameter transformations. From File blocks are ideal for long time vectors since From File blocks do not impact the time required for code generation or compiling. New MAT-files containing From File block data can be created using only MATLAB, or, if you write your own C program that uses functions provided with `libmat.dll`, you can create new signal data without MATLAB. In order to increase the overall flexibility of RSim, slight differences exist in the way data is handled by From File blocks and From Workspace blocks. Theses differences in handling data for From File and From Workspace blocks are summarized in table 2.

Table 2: Comparisons Between From File and From Workspace Behavior Using the RSim Target

From File block	From Workspace block
Length of input signals can change between simulations.	Length of input signals is always the same.
Signal data remains stored in MAT-file.	Signal data placed in generated code requires longer compile time.
Parameter transformations not supported.	Parameter transformations are supported.
Always requires a MAT-file containing signal data to run the RSim simulation.	Data encapsulated in executable, therefore can run simulation without having the MAT-file present.

Portability of RSIM Simulations

Since the RSim Target creates a stand-alone executable, you can copy this executable to another computer for remote execution (e.g., one with the same processor and same operating system). For example, if you are using a PC and a friendly coworker also has a PC, he or she may let you to use idle CPU time at the end of the day. RSim can help by letting you run independent simulations on idle machines.

There are a few things to consider when running an RSim executable on another computer. Assume the other computers do not include MATLAB. In order for RSim to read MAT-files for either new From File block data or new parameter data, the files `libmat.dll` and `libmx.dll` must be available. You can copy these to the machine's working directory or use file sharing and an appropriate path so that these files can be found elsewhere. An easy approach is to simply copy these files to the working directory where you intend to run your RSim simulations.

If you use a model with one or more From File blocks, you also need to copy the MAT-file(s) to your working directory. The same requirements exist for using replacement parameter MAT-files.

Once this has been done, you can write a simple `.bat` file for a PC or write a UNIX script to run your set of simulations. The flags available with RSim executables are as follows.

`-f oldfromfile.mat=newfromfile.mat` (replace From File block data using `newfromfile.mat`)

`-o myoutput.mat` (log simulation results to the file `myoutput.mat`)

`-p myparams.mat` (specify a new parameter .mat file `myparams.mat` which can also including data for From Workspace blocks)

`-s 10.9` (stop the simulation when it reaches t = 10.9)

`-t oldtofile.mat=newtofile.mat` (replace `oldtofile.mat` with `newtofile.mat` for saving data from To File blocks)

`-v` (set verbosity on to see which file(s) are being used as replacements)

Simulation Results

With RSim simulations, logging of simulation results is controlled by menu selections in the Simulink model before code generation takes place. Scope blocks and outports can be connected to signals that you want to view and analyze once the simulation has completed. Be sure to check the appropriate check boxes in the Simulation Parameters dialog box under the Workspace I/O tab. This will allow you to control whether or not signals are saved to a MAT-file. Once the simulation has completed, you can load the simulation results into MATLAB for plotting and analysis of the resulting data.

At the time of code generation, the variable name used to store the time vector and output vector (e.g., y-vector) is selected. If you plan to load simulation results from multiple RSim simulations, rename these variables upon loading data to avoid overwritting these vectors when the next set of simulation results is loaded.

This paper was written by Jim Shefcyk, a member of the Real-Time Workshop development team at The MathWorks, Inc. In this paper it is assumed that you are using the MATLAB Release 11 CD for MATLAB, Simulink and Real-Time Workshop. Please forward any comments, questions, or suggestions on the Rapid Simulation Target, to jims@mathworks.com.
