**Presented at:**
**AIAA Modeling and Simulation Technologies Conference and Exhibit**
**20 - 23 August 2007, Hilton Head, South Carolina**

# The Use of Computing Clusters and Automatic Code Generation to Speed Up Simulation Tasks

Jason R. Ghidella[1], Amory Wakefield[2], Silvina Grad-Freilich[3], Jon Friedman[4] and Vinod Cherian[5]
*The MathWorks, Inc. Natick, MA, 01760*

**This paper studies a number of different techniques that can be used to reduce the amount of time needed to run block diagram simulations. The first is automatic code generation techniques used to create simulation executables from graphical block diagram models. A number of alternative techniques are studied, highlighting increases in simulation speed that can be achieved at the expense of interactivity with the graphical model. This paper will discuss at which stages of modeling and simulation code generation should be considered. The second technique that is studied is the use of computing clusters to distribute a number of simulation runs across a number of processors. With the advent of the multicore processor this technique has become accessible to many more engineers than in the past.**

## I. Introduction

THE use of modeling and simulation has become widespread in the development of embedded systems, driven primarily by the popularity of Model-Based Design[1,2,3]. Migrating to a workflow using Model-Based Design has many benefits, including improving the resulting design, shortening development time and costs, and reducing design errors. Model-Based Design provides insight into system behavior without the need to use physical prototypes of the system, which in turn enables engineers to gain this understanding early in the development process. There are many examples that illustrate the value of using models for the early detection of errors or gaps in the requirements and design, for better understanding of the system behavior, and for evaluating different design scenarios.

A drawback in modeling and simulation is that time must be invested to build the model, and the time required can be significant when that model is written in a programming language such as C. This has driven the use of commercial off-the-shelf (COTS) software in the development and simulation of models for embedded systems, especially when the software can provide specialized, domain-specific functionality, because it can shorten the modeling and design process by making it easier to develop high fidelity models of embedded systems[4,5,6].

As COTS software has made it easier to implement Model-Based Design for embedded systems, the use and reuse of those models throughout an organization continues to increase. Organizations are also finding new applications and ways to take advantage of the intellectual property that is contained within these models. Many of these applications require simulating the model multiple times for various purposes.

For example, design exploration studies, Monte Carlo analysis, robustness testing, parameter sweeps, requirements analysis, bit error rate (BER) calculations, and other verification and validation activities all rely on multiple simulation iterations. Running the model many times on different test cases provides insight to the engineer for refining and improving their design, as well as verifying the design is meeting the system requirements. Each of these runs may take hours to complete. Depending upon the complexity of the model, simulation time can become a critical bottleneck in the development process. One way to speed up verification tasks is to automate test

[1] Manager, Technical Marketing, 3 Apple Hill Drive, Natick, MA, 01760, AIAA member.
[2] Technical Marketing Manager, 3 Apple Hill Drive, Natick, MA, 01760.
[3] Manager, Parallel Computing Marketing, 3 Apple Hill Drive, Natick, MA, 01760.
[4] Manager, Aerospace, Defense and Automotive Marketing, 3 Apple Hill Drive, Natick, MA, 01760.
[5] Technical Marketing Specialist, 3 Apple Hill Drive, Natick, MA, 01760.

cases. Storing these test cases independently of the model allows them to be easily reused as the model is refined over time. Alone, this does not fully resolve the time bottleneck, but in combination with other techniques, it can greatly reduce the time spent on multiple simulations.

Other activities such as algorithm prototyping, processor-in-the-loop (PIL), or hardware-in-the-loop (HIL) simulations rely on techniques to automatically generate code from the model. This code can then be targeted to microprocessors, DSPs or FPGAs that interact with prototype hardware in real time[7,8]. Expanding the use of code generation for simulation acceleration can also help alleviate the simulation speed bottleneck discussed above; however, it alone cannot completely address the need for faster simulations.

This has driven a call for the ability to run simulations in a high-performance computing (HPC) environment. In the past, HPC was available mainly to government agencies and large research laboratories, because only these groups had the means to purchase supercomputers. Today, most supercomputers have been replaced by COTS computer clusters that provide affordable, high-performance, distributed environments. In addition, the number of available clusters is growing rapidly. HPC is now at all-time high of $10 billion in hardware revenue in 2006 and a growth of more than 20% over last 4 years[9].

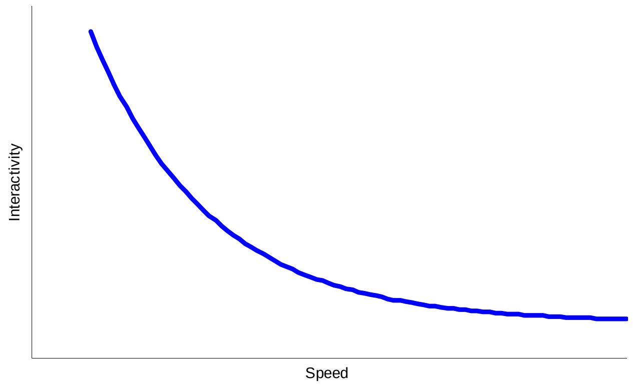


**Figure 1. Simulation speed relationship to model interactivity. In general, simulation speed increases as model interactivity decreases.**

With the increased availability of multicore and multiprocessor computers, and the growth of computing clusters, access to vast CPU processing power is now within the reach of many engineers. By combining computing clusters with code generation technology, models can be easily built with COTS software, and simulation tasks can be completed orders of magnitude faster.

In this paper, we will discuss multiple techniques to reduce the time needed to complete Model-Based Design tasks with the COTS software tools, MATLAB®[10] and Simulink®[11] from The MathWorks. This will include the use of HPC clusters, code generation technology, and test reuse. Examples and benchmarks will be discussed that demonstrate the typical increase in performance that can be achieved.

## II. Using Code Generation for Simulation Acceleration

There are many reasons a model may take a long time to simulate, some of which are described in the previous section. The rest of this paper will use examples based on Simulink models, and some specific techniques and tools developed to speed up the simulation time of Simulink models. The Simulink User's Guide[10] discusses some of the common reasons why a simulation can take a long time to run, from having a model that contains an algebraic loop, to setting the maximum step size too small, or simply having too many scopes in the model.

When these common reasons for simulation slow down have been investigated and corrected and the model still simulates slower than desired, the use of automatically generated C code to speed up the simulation should be considered. However, converting the graphical model into an executable loses many of the benefits of modeling the system in a COTS block diagram software tool. With this in mind, an engineer may wonder if modeling should be done in a programming language like C from the start. When compared to a programming or scripting language, a block diagram tool can save large amounts of time in describing and defining a system. Additionally, even if describing the system is easy to do in C, solving that system through time, and debugging the C code is a lot more difficult than debugging a graphical model.

Because it is easier to build and debug models in a graphical block diagram, using C code to accelerate simulation is more effective later in the development process. A good point to consider using C is when engineers start using, rather than building, the model. That is, when they start simulating the model many times without changing its structure. Some tasks that require this include: investigating the full design space, robustness testing to ensure the design can deal with uncertainties, and verification and validation tasks to ensure the implementation does what is required and is the right design. At these times, the structure of the model is well defined, and the only changes that will be made to the model are typically input signals and model parameters.

The reason C code should only be considered once the behavior of the system has been defined and validated is illustrated in Figure 1. In general, as the simulation speed increases, the user will have less flexibility in running and interacting with the simulation. For example, debugging tools will not be available or some modeling semantics may not be compatible. Finally, it takes time to generate the C code from the model. So, when engineers are constructing a model or changing the model structure as the model is validated, the code will have to be rebuilt not only each time a simulation occurs but also if they simply update the diagram to ensure all the signal data types are consistent in the model. This can consume more time than the savings gained from running the simulation at compiled C code speeds.

This means that as an engineer builds a model, using the default simulation mode (Normal) will be the most efficient. When the engineer then wants to use that model, switching the simulation mode to one based on code generation can then be a better approach if the simulation is not running fast enough. At this stage the model is well defined and the focus is now on having the simulations run as fast as they can.

The value of working with Simulink is that engineers can realize all the benefits of a COTS graphical block diagram tool to build system models quickly and then use a simulation mode that automatically creates a compiled executable of that model so that they can achieve compiled C code simulation speeds as well without having to leave the graphical environment.

## A. Simulink Simulation Acceleration Modes

Simulink offers four simulation modes (Normal, Accelerator, Rapid Accelerator, and External), to help build, execute, and debug models. Two of these simulation modes, Accelerator, and Rapid Accelerator, use code generation technology to generate a compiled executable of the model. The underlying C code for the model is not accessible by the user for either of the executables generated by the Accelerator or Rapid Accelerator modes.

While running accelerated simulations, the user is still able to interact with the Simulink model, such as view signals on scopes, but this interaction does decrease as the user moves from the default, Normal mode, to Accelerator mode, to Rapid Accelerator mode. For example, if the Simulink model contains an algebraic loop (where the input of a block with direct feedthrough is driven by the output of the same block), neither acceleration mode will work. If the model contains blocks that cannot generate C code, for example a Transport Delay or MATLAB Fcn block, the Accelerator mode will interpret those blocks at reduced speed, whereas the Rapid Accelerator mode will not work. The Accelerator mode is compatible with the Simulink Profiler and Debugger, whereas the Rapid Accelerator mode is not.

Users do not need to set up the code generation process to use these simulation modes. They are shielded from this because all they need to do is to select the mode itself. Once the simulation is started, a compiled executable is then generated, if needed, before the simulation starts running. The only option the user selects is the compiler optimization level: turned off for faster build times or turned on for faster simulation speeds at the expense of longer build times.

The final simulation mode, External mode, forms a communications link between a model running in Simulink and code from that model running on a target system. In External mode, the Simulink model is the user interface to the external code, enabling users to download modifications to block parameters to the external code, and view signals from the external code in real time.

The Accelerator mode is the more general purpose of the two simulation acceleration modes. It creates a compiled executable for the parts of the model that can be compiled into C code and uses the interpreted mode for the other blocks. It provides a high level of interactivity so that the user can use this mode even during the model construction and debugging phases if needed. The compiled executable is solved through time using Simulink solvers. This compiled executable runs in the same process as MATLAB and Simulink. Block parameters in the model can be changed while the simulation is running, and scopes and animation display as the simulation runs.

Extensive analysis is performed to avoid unnecessary rebuilding of code for simple changes to the model such as changing the simulation stop time or the solver type.

The Rapid Accelerator mode is more restrictive and will only work with models containing blocks that can compile into C code. The Rapid Accelerator mode creates a standalone executable from a model that also includes the specific solver methods for that model. Simulink does not solve the model through time, as it does in the Accelerator mode. The standalone executable created by the Rapid Accelerator mode runs as a process separate from MATLAB and Simulink, so, if a second processing core is available, it will take advantage of that core. Significant speed increases in simulation time can be achieved with this mode. Block parameters can be changed without rebuilding only if they have been defined as tunable parameters. Changes to the solver type or stop time, however, will necessitate a rebuild of the code. In general, the Rapid Accelerator mode will require rebuilds more frequently than the Accelerator mode.

## B. Simulation Target

If engineers need access to the underlying code that is generated from the model, or they want to distribute that code on different targets, they can do so using Real-Time Workshop®[12] from The MathWorks. Real-Time Workshop generates ANSI/ISO C/C++ code from Simulink models. Using target template files engineers can specify on which environment the code will run. One target template that is specific to simulation acceleration is the Rapid Simulation target (RSim).

RSim is a separate executable from Simulink that is executed from a DOS command line (on Windows platforms), thus requiring scripts to get data into and out of the executable. RSim is compatible with External mode, so engineers are still able to interact with the executable through the Simulink model. This, however, is not as convenient as the Rapid Accelerator mode that has automated the interaction between the simulation executable and the model.

RSim has a slight speed advantage over the Rapid Accelerator mode in running simulations, because there is no check to see if the model is up to date. RSim starts simulating immediately, whereas the Rapid Accelerator mode first checks to see if the model has changed before the simulation begins.

## C. Benchmarking results

Simulation acceleration can be difficult to quantify, given that the factors that can slow down a particular simulation could stem from a wide range of issues. Some of these are described in the previous sections. Because there are so many factors that need to be taken into account that can affect overall simulation speed, it is not possible to define a benchmark that will be representative of every situation.

In this paper, we do not attempt to present an exhaustive set of examples to show how these factors can affect overall simulation speed using different techniques. Instead, one example is presented using a model of a fault detection, isolation, and recovery (FDIR) subsystem for a redundant elevator control system that has been described in detail in previous work[13,14]. Figure 2 shows a view of the FDIR subsystem model, which provides a means to study, in simulation, the effects of one or multiple failures in the actuation system of the elevators in an aircraft. The model contains 897 Simulink blocks and a comparable amount of Stateflow® logic and is representative of a component of a larger complex vehicle system model.

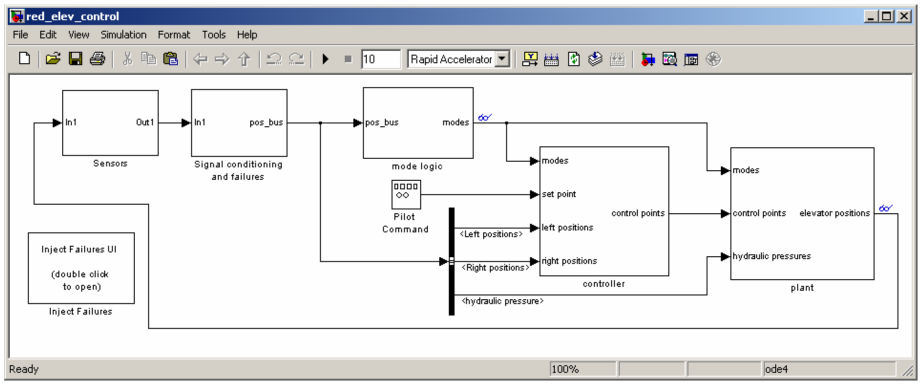


**Figure 2. View of the top level model used to benchmark the performance of simulation modes.**

Figure 3 shows the clock time taken to run 16 different fault scenarios, each of which are simulating 1000 seconds of the dynamics of the FDIR subsystem model. The height of the blue bars represents the time needed to complete the simulation tasks using different simulation modes and RSim. The height of the red bars represents the additional overhead required to execute the simulation. This overhead is composed of such tasks as update diagram, code generation, code compilation and resource allocation before executing the simulation.

As expected, Normal mode is the slowest, requiring 1232 seconds to simulate 1000 seconds of simulation time for the 16 different fault scenarios. This is because Normal mode does not compile any blocks. The Accelerator mode is faster, taking 812 seconds for the same simulation tasks. The speed up is provided by using precompiled code. The red stack representing overhead time in Figure 3 includes the one-time cost of generating the compiled code and post-processing of the compiled code. At 166 seconds of execution time, the Rapid Accelerator mode provides further increase in speed. Note, however, that the overhead is now higher, because the compiled code has to pass stricter checks between simulation runs to ensure model fidelity. RSim is the fastest, taking just 75 seconds to complete the simulation tasks.

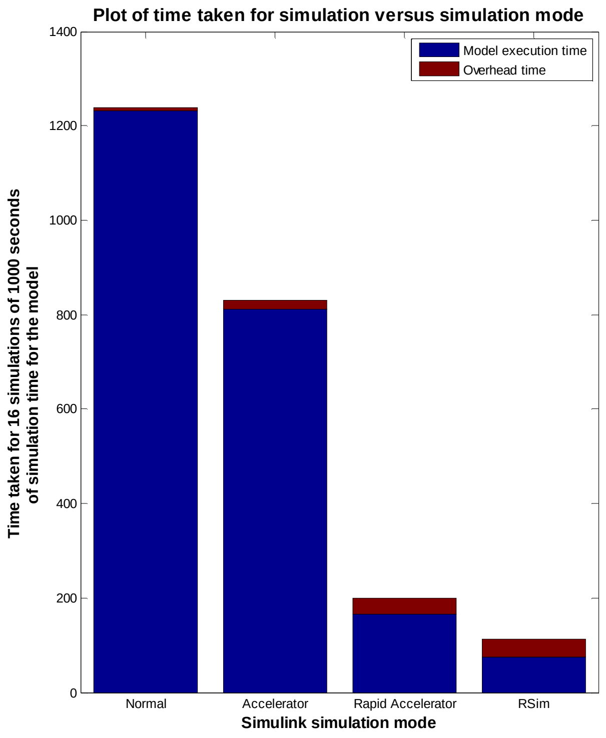


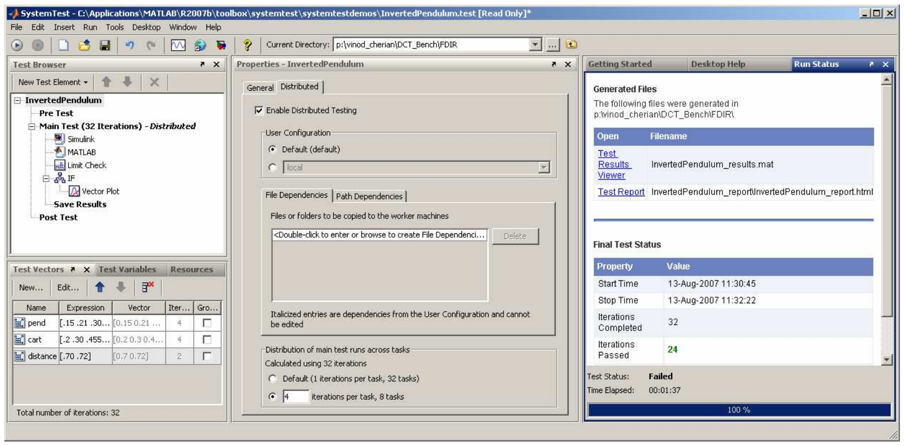
**Figure 3. Plot showing the time taken for simulating the task using different Simulink simulation modes.**

## III. Using a Computing Cluster to Speed Up Simulations

Code generation techniques can help decrease the time require for one simulation. Today's complex systems and safety-critical applications require evaluating or testing multiple scenarios of a design. This is one reason many organizations have turned to Model-Based Design. Model-Based Design makes it much easier and more cost-effective to iterate designs, verify and validate designs, and test designs. It enables engineers to wait until the model is complete and finalized before building costly, long lead-time prototypes and using limited, expensive lab time. Engineers can use hardware and lab testing only to verify that the simulation results match the real system, rather than using it as a design step. Simulations also allow engineers to model events or conditions that may be difficult to recreate in a lab environment, such as extreme temperatures or crash testing.

There are many techniques that engineers use to simulate their designs over the large parameter spaces required by design exploration studies, robustness testing, or BER. Design of Experiments (DOE) and Monte Carlo analysis are two techniques that require evaluating multiple scenarios. In fact, a major benefit of using Design of Experiments techniques is to limit the number of scenarios that need evaluation. This is because building, or even modeling, all possible scenarios is impractical—it takes too long. To reduce evaluation or testing time even further requires more innovation. The ability to run simulations in a HPC environment is one approach that was once prohibitively expensive to implement due to the cost of installing and maintaining sufficient computing power. With the expansion of multicore and multiprocessor computers, and the growth of computing clusters, vast CPU processing power is already installed in many organizations' sites. Taking advantage of that computing power using the same tools installed on an engineer's desktop is not always so straightforward. In fact, the difficulty in setting up distributed applications is still a barrier to tapping the processing power in one computer, let alone a cluster of perhaps hundreds. Yet, taking advantage of this resource is a critical step to speeding up simulations, as well as increasing the number of simulations that can be reasonably run.

Some problems are called distributed or coarse-grained because they can be segmented easily to run on several nodes without communication, shared data, or synchronization points between the nodes. Monte Carlo simulations fall into this category, and thus are an obvious candidate to run on computing clusters. Getting an individual's simulations to the cluster or onto multiple processors, particularly from within a block diagram software tool, is not always as obvious. One possibility is to compile the model into an executable and then script the parameter variations, communication with a scheduler, and job creation. This can be tedious and often requires help from a cluster administrator to customize a particular simulation setup. The MathWorks has responded to this need by providing distributed computing tools that make it possible to distribute complete Simulink models for execution in a cluster or in a multicore or multiprocessor computer without the need to write any lines of code.



InvertedPendulum
- Pre Test
- Main Test (32 Iterations) - *Distributed*
  - Simulink
  - MATLAB
  - Limit Check
  - IF
    - Vector Plot
  - Save Results
- Post Test

| Name | Expression | Vector | Iter... | Gro... |
|---|---|---|---|---|
| pend | [.15 .21 .30... | [0.15 0.21 ... | 4 | ☐ |
| cart | [.2 .30 .455... | [0.2 0.3 0.4... | 4 | ☐ |
| distance | [.70 .72] | [0.7 0.72] | 2 | ☐ |

Total number of iterations: 32

☑ Enable Distributed Testing

User Configuration
(●) Default (default)
( ) local

Files or folders to be copied to the worker machines
<Double-click to enter or browse to create File Dependenci...

Italicized entries are dependencies from the User Configuration and cannot be edited

Distribution of main test runs across tasks
Calculated using 32 iterations
( ) Default (1 iterations per task, 32 tasks)
(●) 4 iterations per task, 8 tasks

**Generated Files**
The following files were generated in p:\vinod_cherian\DCT_Bench\FDIR\

| Open | Filename |
|---|---|
| Test Results Viewer | InvertedPendulum_results.mat |
| Test Report | InvertedPendulum_report\InvertedPendulum_report.html |

**Final Test Status**

| Property | Value |
|---|---|
| Start Time | 13-Aug-2007 11:30:45 |
| Stop Time | 13-Aug-2007 11:32:22 |
| Iterations Completed | 32 |
| Iterations Passed | 24 |

Test Status: **Failed**
Time Elapsed: 00:01:37
100 %

**Figure 4. The SystemTest desktop showing an example of using multiple processors by selecting a checkbox in the GUI.**

SystemTest[15] and Distributed Computing Toolbox[16] make it possible to set up a series of parameters in a Graphical User Interface (GUI) to vary in a Simulink model for Monte Carlo or DOE and then define it as a job to be run on multiple processors. Launching the job from SystemTest requires only a checkbox, as shown in Figure 4.

MATLAB® Distributed Computing Engine[17] schedules and executes the job on available workers, which are MATLAB processes running on a cluster. A system or cluster administrator can setup a configuration file and make it available to all who want to use the cluster. That file contains information about the particular scheduler in the cluster and the shared directories that may be needed. Each worker requires only the MATLAB engine license; separate toolbox and blockset licenses are not required. The distributed computing tools implement dynamic licensing, which enables the workers to run software that uses any eligible MATLAB toolboxes or Simulink blocksets for which users are licensed on the computer sending the job.

## IV. Combining Code Generation with a Computing Cluster

For maximum benefit, both code generation techniques and a computing cluster can be used. SystemTest and Distributed Computing Toolbox can run a Simulink model in any of the simulation modes. Thus, users can take advantage of the fastest speed for each single simulation run, as well as distributing the faster simulations across multiple machines.

Figure 5 demonstrates the speed improvement achieved by distributing the tasks used in Section II C of this paper on a computing cluster of up to eight machines. Table 1 lists the performance improvement as a factor of the base rate of one machine running the task. For example, executing the simulation in the Normal mode on two workers produced a 2.0x speed improvement over a Normal mode simulation on one machine. Similarly,

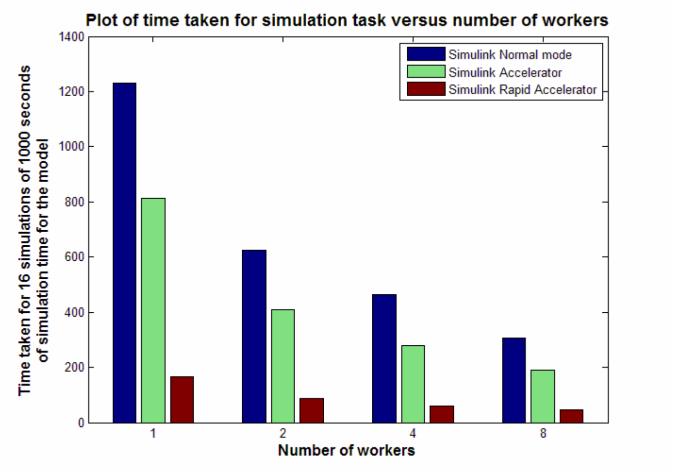


**Figure 5. Plot of speed-up achieved using different simulation modes over multiple processors.**

executing a Rapid Acceleration mode simulation of the task on eight workers produced a 26x speed improvement over a Normal mode simulation on a single machine.

As can be seen in Figure 5 and Table 1, the speed improvement is not linearly dependent on the number of workers. This is because distributing the tasks adds its own overhead, namely, copying over the files from the development machine to each machine on the cluster, transmitting the input and output data between the workers and the job manager – in this case the MathWorks job manager provided with the MATLAB Distributed Computing Engine, and the network latency. In general, total simulation time should be significantly longer than this overhead to achieve speed benefits from distributing tasks.

| Number of cores / Simulation mode | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| Simulink Normal mode | 1.0x | 2.0x | 2.7x | 4.0x |
| Simulink Accelerator mode | 1.5x | 3.0x | 4.4x | 6.5x |
| Simulink Rapid Accelerator mode | 7.4x | 14x | 21x | 26x |

**Table 1: Performance gains to be had from different simulation modes and distributing the simulation on a cluster.**

In the example, the Rapid Accelerator mode shows decreasing rates of improvement as the tasks are distributed among four and eight workers. The effect is less pronounced in the case of the Normal mode simulations. This is different from the results in Figure 3, where the example has eliminated the overhead from the model build process for the Accelerator and Rapid Accelerator modes. This is because the Accelerator and Rapid Accelerator modes do not need to rebuild the code on the single processor; once the model has been built, unless there is a change in the model structure, it can just rerun the same executable for each simulation task. When the simulation is distributed, each worker must build the code once. The results obtained here are specific to the model and the example cases used.

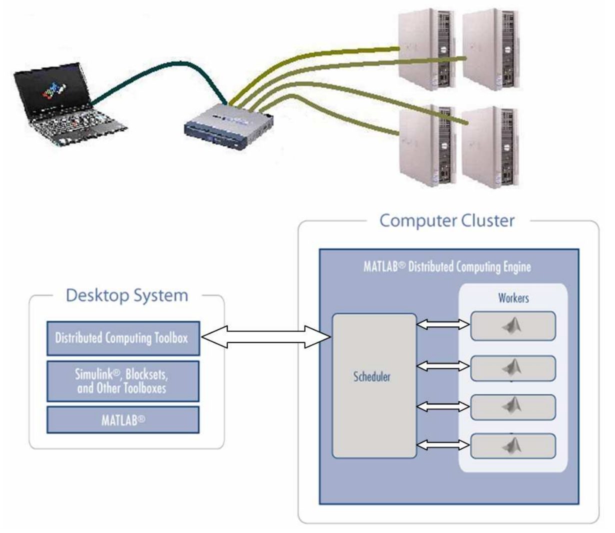

Computer Cluster
MATLAB® Distributed Computing Engine
Workers
Scheduler
Desktop System
Distributed Computing Toolbox
Simulink®, Blocksets, and Other Toolboxes
MATLAB®

**Figure 6. Network topology and schematic of the cluster used to perform tests in the distributed simulation.**

The dedicated cluster of machines used for this example consists of four identical Dell OptiPlex SX280s[18] with dual core processors running at 2.8GHz and with 1GB of shared memory on each machine. These machines are connected to each other via a dedicated router and subnetwork. The router is also connected via a larger network to a dual core, 1.83GHz IBM T60 laptop with 1GB of memory that is used to prototype the code and run the scheduler. All the machines on the cluster were running the 32-bit Windows XP operating system. Figure 6 shows the topology of the computing cluster. Distributed Computing Toolbox and MATLAB Distributed Computing Engine support other platforms and heterogeneous computing clusters with different machines running different operating systems. Different network topology or machine architecture could make a difference to the results obtained, in addition to the other factors discussed in the paper.

RSim was not investigated in this section as it involves the manual generation of target executables that are specific to platform architecture and operating system. RSim is expected to produce similar results to the Rapid Accelerator mode as more workers are added to the cluster.

## V. Conclusion

There are many reasons engineers need to run simulations faster. This paper has presented several methods that can be used to reduce the time taken to run simulations. The choice of techniques depends heavily on the user's specific task and should take into consideration such things as debug needs, network communication overhead, total simulation time, compile time (for code generation techniques), access to processing power and maturity of the model.

The Simulink example shown in the paper proves that tools from The MathWorks can be successful in achieving more than 20x improvement in total time spent running a simulation using only eight processors and a relatively short number of simulation tasks. It is possible to see how beneficial these techniques could be when a single simulation takes hours to run. This paper has addressed several methods by which faster speeds can be achieved running Simulink models. As HPC clusters and multicore computers become more prevalent, these improvements will be achievable by all engineers.

## References

[1] Barnard, P., "Graphical Techniques for Aircraft Dynamic Model Development," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Providence, Rhode Island, Aug. 2004, CD-ROM.
[2] Aberg, R., and Gage, S., "Strategy for Successful Enterprise-Wide Modeling and Simulation with COTS Software," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Providence, Rhode Island, Aug. 2004, CD-ROM, ID: 2004-4929.
[3] Krasner, J., "Model-Based Design and Beyond: Solutions for Today's Embedded Systems Requirements," Embedded Market Forecasters, American Technology International, Framingham, MA, January 2004.
[4] Wood, G. D., and Kennedy, D. C., "Simulating Mechanical Systems in Simulink® and SimMechanics", Technical Report 91124v00, The MathWorks, Inc., Natick, MA, 2003.
[5] Denery, T., Ghidella, J. R., Mosterman, P. J., and Shenoy, R., "Creating Flight Simulator Landing Gear Models Using Multidomain Modeling Tools", *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Keystone, Colorado, Aug. 2006, CD-ROM, ID: 2006-6821.
[6] Popinchalk, S., Glass, J., Shenoy, R., and Aberg, R, "Working in Teams: Modeling and Control Design within a Single Software Environment", *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Hilton Head, South Carolina, Aug. 2007, CD-ROM.
[7] Hodge, G., Ye, J., and Stuart, W., "Multi-Target Modelling for Embedded Software Development for Automotive Applications," *2004 SAE World Congress*, 2004-01-0269, SAE International, Detroit, MI, March, 2004.
[8] Erkkinen, T., "Multiple Purpose Automatic Code Generation", *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Hilton Head, South Carolina, Aug. 2007, CD-ROM.
[9] IDC HPC Briefing at the 2007 International Supercomputing Conference
[10] "Getting Started with MATLAB®", The MathWorks, Natick, MA, September 2007.
[11] "Using Simulink®", The MathWorks, Natick, MA, September 2007.
[12] "Real-Time Workshop® User's Guide", The MathWorks, Natick, MA, September 2007.
[13] Mosterman, P. J., and Ghidella, J., "Model Reuse for the Training of Fault Scenarios in Aerospace," *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, Providence, RI, Aug. 2004, CD-ROM, ID: 2004-4931.
[14] Ghidella, J., and Mosterman, P., J., "Requirements-Based Testing in Aircraft Control Design", ," *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, San Francisco, CA, Aug. 2005, CD-ROM, ID: 2005-5886.
[15] "SystemTest User's Guide", The MathWorks, Natick, MA, September 2007.
[16] "Distributed Computing Toolbox User's Guide", The MathWorks, Natick, MA, September 2007.
[17] "MATLAB® Distributed Computing Engine System Administrator's Guide", The MathWorks, Natick, MA, September 2007.
[18] http://www.dell.com/html/us/products/optiplex/sx280.html