

Battery Pack Modeling, Simulation, and Deployment on a Multicore Real Time Target

Javier Gazzarri, Nishant Shrivastava, Robyn Jackey, and Craig Borghesani
MathWorks

ABSTRACT

Battery Management System (BMS) design is a complex task requiring sophisticated models that mimic the electrochemical behavior of the battery cell under a variety of operating conditions. Equivalent circuits are well-suited for this task because they offer a balance between fidelity and simulation speed, their parameters reflect direct experimental observations, and they are scalable. Scalability is particularly important at the real time simulation stage, where a model of the battery pack runs on a real-time simulator that is physically connected to the peripheral hardware in charge of monitoring and control. With modern battery systems comprising hundreds of cells, it is important to employ a modeling and simulation approach that is capable of handling numerous simultaneous instances of the basic unit cell while maintaining real time performance.

In previous publications we presented a technique for the creation of a battery cell model that contains the electrochemical fingerprints of a battery cell based on equivalent circuit model fitting to experimental data. In this work we extend our previous model to represent a battery pack, featuring cell creation, placement, and connection using automation scripts, thus facilitating the design of packs of arbitrary size and electrical topology. In addition, we present an assessment of model partitioning schemes for real time execution on multicore targets to ensure efficient use of hardware resources, a balanced computational load, and a study of the potential impact of the calculation latencies inherent to distributed systems on solver accuracy. Prior to C code generation for real time execution, a model profiler assesses the model partitioning and helps determine the multicore configuration that results in the lowest average turnaround time, the time elapsed between task start and finish.

The resulting model is useful in the generation of multiple operating scenarios of interest in the design of charging, balancing, and safety related procedures.

CITATION: Gazzarri, J., Shrivastava, N., Jackey, R., and Borghesani, C., "Battery Pack Modeling, Simulation, and Deployment on a Multicore Real Time Target," *SAE Int. J. Aerosp.* 7(2):2014, doi:10.4271/2014-01-2217.

INTRODUCTION

Many battery applications, especially those of high power levels, require the series or parallel connection of several unit battery cells, with a topology dictated by the electrical characteristics of the load. As the battery pack size increases, so does the computational demand on the solver for the simulation of the system. This problem becomes more acute if the model needs to run in real-time for hardware-in-the-loop (HIL) testing. Modern computing hardware offers capabilities such as parallel execution that can increase simulation speed, but it is generally not easy to fully take advantage of these capabilities, largely due to the significant challenges posed by multithreading the model execution process.

Real time execution on multicore targets would appear to be suitable for battery pack simulation because the repetitive nature of a battery pack suggests a relatively straightforward way to evenly partition the model. This apparent simplicity notwithstanding, the implications of partitioning a model in terms of its data transfer latencies on solver accuracy is a more difficult problem to solve.

Desktop simulation is a natural first step before executing generated code on a HIL platform. This work presents a technique in which the designer uses desktop simulation to determine optimum partitions of the model prior to code generation and execution on target hardware. This step enables the designer to achieve a balanced computational load distribution across the multithreaded HIL application.

A second goal is to provide a scalable methodology capable of supporting any number of battery cell components configured in series or parallel. A MATLAB script creates, places, and connects each model component, including battery cell blocks and electrical components for the battery pack load, and partitions the model.

MODEL PARTITIONING

As system complexity increases, so too does the computational demand placed on simulation hardware. When modeling and simulating a battery pack, thermal management and charge balancing require a model that encompasses the entire battery pack and that can be simulated in real time.

Real time capability is an essential requirement for HIL testing. For large models, real time performance can be difficult to achieve, especially if the system contains components that are numerically stiff, comprising subsystems with diverse time constants. One option for achieving real time simulation of large models is to distribute computational load by partitioning for multicore execution. With this approach, some tasks are executed concurrently, decreasing the execution time required by the solver to advance each simulation step.

In Simulink, the user configures a model for on-target concurrency by assigning separate *tasks* to different parts of the model at the top level. Each task requires an associated sampling time that must be an integer multiple of the top-level model step size. This partitioning is done irrespective of the target hardware specifications, with a scheduler assigning the tasks to CPU cores in an optimal way without the need of user intervention. This detachment of model development from hardware mapping makes this approach scalable and easy to employ with increasingly powerful targets. In Figure 1, model blocks are assigned tasks that the Simulink scheduler later uses to generate multithreaded C code for on-target execution.

Although concurrency is only possible at the on-target execution stage, Simulink offers a profiling tool to assess the performance of the partitioned model on the desktop. This tool simulates the effects of model partitioning in terms of computational load balance and shows quantitative information on maximum and average turnaround times for the execution of each task. We use this tool to assess the partitioning strategy of a cell pack.

The often homogeneous architecture of a battery pack makes it suitable for concurrency since the model can be partitioned into similar units that should result in a balanced load on the computing hardware. It is important, however, to validate the partitioned model to ensure that delays and data transfer latencies introduced by the partitioning do not undercut solver performance. The first part of the Results section describes this validation step in detail. Subsequent parts of the Results section describe the process used to choose the optimal partitioning scheme for a battery pack architecture of eight lithium ion cells in series.

RESULTS

Baseline Model

The baseline model used to validate partitioning strategies consists of eight cells in series. Each battery cell block contains an equivalent circuit with state-of-charge and temperature dependent parameters, as described in [1,2,3]. Figure 2 shows the benchmark model to be compared against partitioned models. Battery cell, voltage sensor, current source, and heat transfer blocks are part of the Simscape library, and they are interconnected forming a physical (acausal) network. In this simulation, a sequence of 25A discharge pulses partially drains the initially fully charged battery pack during one hour.

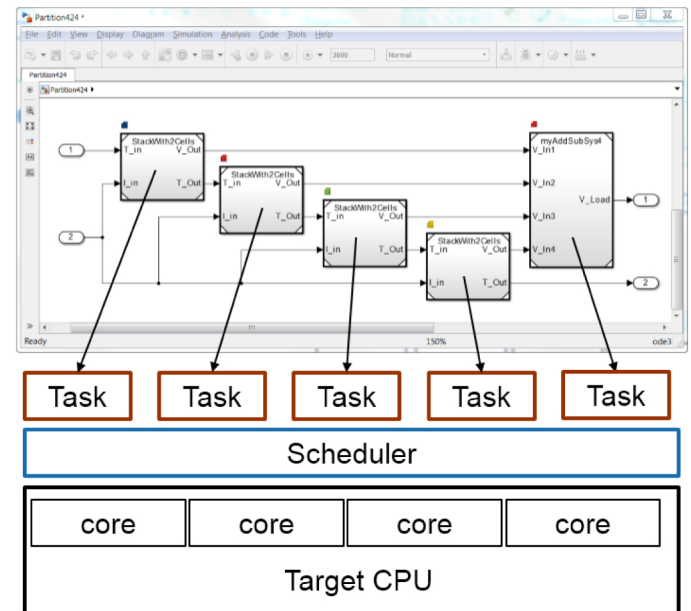


Figure 1. Partitioning schematic for concurrent execution in Simulink. The user assigns different tasks to different parts of the model and a scheduler finds an optimal hardware distribution of the computational load [4].

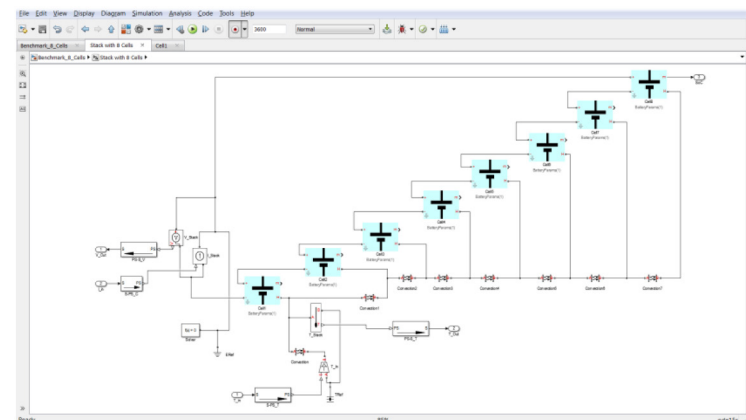


Figure 2. Baseline model with eight lithium ion battery cells in series. Thermal connection blocks model convective heat transfer between adjacent cells. The baseline model uses the variable step implicit solver (ode15s) recommended for numerically stiff systems.

Inputs to the model are the discharge current and the ambient temperature, and outputs are the pack voltage and temperature. The baseline model uses a variable-step solver for maximum accuracy. Subsequent versions of the model are configured with a fixed-step solver to enable real-time simulation. It is important to compare the variable step results and the fixed step results to identify any possible losses in accuracy, particularly during rapid changes in voltage and current.

Figure 3 shows the voltage (V), temperature ($^{\circ}\text{C}$), current (A), and state of charge (or SoC) traces as a result of the electrical and thermal stimuli.

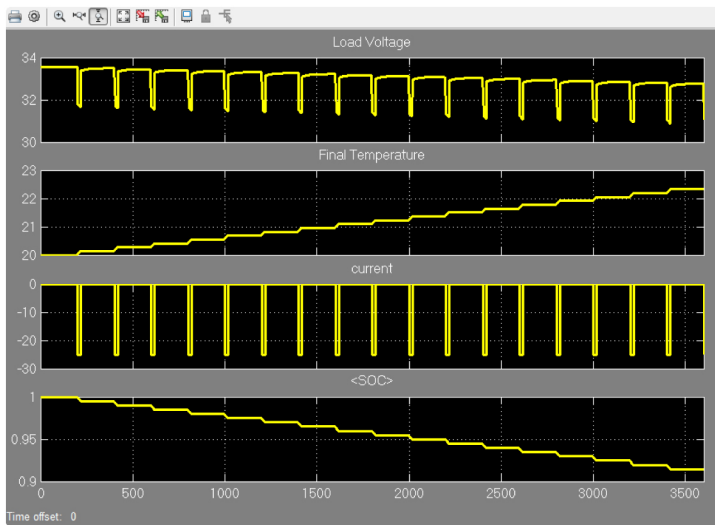


Figure 3. Pack voltage (V), temperature ($^{\circ}\text{C}$), discharge current (A), and SoC (-) for the benchmark model.

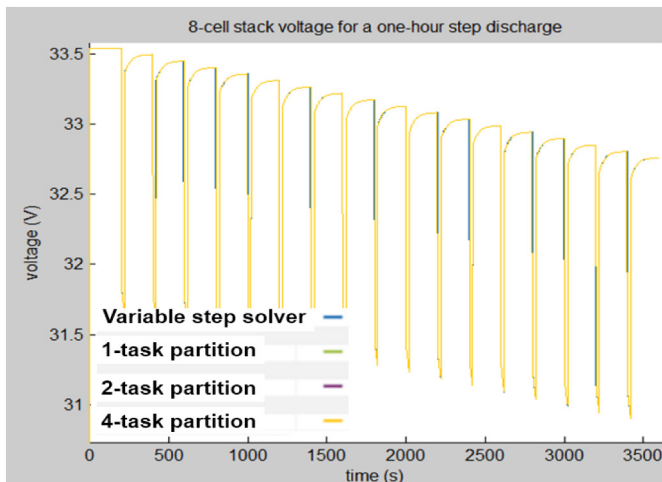


Figure 4. Eight-cell pack voltage response for the baseline simulation and three partitioning schemes: 1) No partition with variable step solver, 2) No partition fixed step solver, 3) Partition in two tasks with four cells each, and 4) Partition in four tasks with two cells each.

Figure 4 shows the voltage response of all partitioned schemes superimposed on the baseline results. The similarity of the results validates the three partitioned versions of the model:

- 1). 1-task partition: One task for the entire eight cell pack. Fixed step solver
- 2). 2-task partition: Two-task partition with four cells each in each task. Fixed step solver
- 3). 4-task partition: Four-task partition with two cells in each task. Fixed step solver

Partitioning Procedure

For a given battery pack size, several partitioning schemes can be conceived for real time concurrent execution. A four-core real time computer suggests, for example, a partitioning of four units with two cells each. When partitioning, it is important to take into consideration the performance and accuracy implications of imposing execution boundaries on the model, since this arrangement also imposes potential data transfer latencies between any two adjacent partitions. This compromise between execution time improvement and latency management lends itself to an optimization scheme.

The automated workflow of model creation streamlines the task of exploring for optimal partitioning schemes. The workflow enables the user to evaluate several configurations and to assess the benefits of multicore computing. The user can also evaluate trade-offs in performance imposed by scheduler overhead and the solver performance implications of inter-task data transfer latencies. Simulink provides a profiling tool to perform this trade-off evaluation.

To support an automated and convenient method for exploring different partitioning architectures, we create the battery pack model using MATLAB scripts. This approach enables the creation of a battery pack model and its partitioning into tasks with a simple call to a function. The script combines unit blocks according to arguments provided to it. The base function is named `CE_CreateModel` and it has seven arguments.

```
CE_CreateModel(ModelName, n_Stacks,
               n_Cells, n_Tasks, base_rate, f, choice)
```

The arguments to this function are: the desired model name, the number of stacks into which the battery pack will be divided, the number of cells in series inside each stack, the number of tasks into which the model is to be partitioned, the top-level solver sampling rate, an integer factor that multiplies the base sample rate if any part of the model needs to be solved at a slower frequency, and a choice of desktop simulation or real time simulation.