2009-01-0268

Optimal Scheduling in Graphical Modeling Environments

Michael Burke
The MathWorks

ABSTRACT

Methods for controlling execution order in traditional text-based languages such as C and Fortran are well established. The transition to graphical programs has revealed some of the hidden issues inherent in any scheduling routine, specifically data dependency and data protection (in multirate systems). Graphical programming languages provided built-in diagnostics that allow users to analyze the data dependencies to develop optimal schedules from a data propagation perspective. This paper examines one heuristic that can be used to develop an optimal schedule for an arbitrary model.

INTRODUCTION

The use of computer-based graphical modeling languages to model control systems has been steadily increasing since the early 1980s. Use of graphical modeling languages is growing, in part, because the graphical representation closely models traditional control design methodologies, making it a favorite tool among control engineers. With the introduction of code generation technologies in the late 1990s, the use of these languages expanded beyond control engineering into the software engineering domain.

Entry into the software engineering domain highlighted the importance of controlling the execution order of the graphical components. Unlike in traditional programming languages such as C or Fortran, where the execution of components (e.g., functions) is explicitly set by a scheduling function, graphical components depend on the connection between components. The graphical language attempts to determine the correct execution order (sorted order) based on the connectivity of the system (see Figure 1).

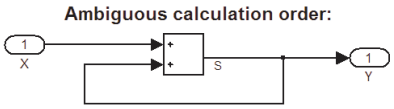

Figure 1. Ambiguous execution order example.

This paper uses Simulink as the reference graphical modeling language. Simulink takes a directed graph approach to determining the execution order of a system. When a system is assembled in such a way that the execution order is ambiguous, the tool infrastructure provides diagnostic error messages (see Figure 2).

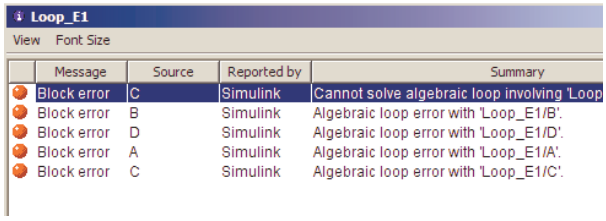

Figure 2. Example error message.

The error messages can be interpreted into the node tree of a data dependency graph. Also, the proper location for resolving the loop can be determined by observing where the flow direction of the graph changes (from input to output or from output to input).

OBJECTIVE - This paper shows how to resolve execution order ambiguity in graphical models by using a simple heuristic based on directed graph methodologies. The resulting model is optimized from the scheduling perspective because it will use the minimum number of loop-breaking blocks (Unit Delay blocks) to resolve the loops. In addition, by following the heuristic method, the end user will have a better understanding of the interdependency of his or her system.

OVERVIEW OF DATA DEPENECIES

Data dependency is the requirement that for any calculation, all the values on the right-hand side (RHS)

of an equation are known prior to starting the calculation. The value on the left-hand (LHS) side is dependent on the values on the right hand side. In a subsystem context, the concept of calculation order is equivalent to execution order of the subsystems.

$$LHS_n = f\,(RHS_n)$$

The C language does not prevent users from writing equations in which the LHS is assigned before the RHS. This means that old or noninitialized data can be used, which can have unexpected or incorrect results.

$$LHS_n = f\,(RHS_{n-1})$$

$$LHS_n = f\,(??)$$

Graphical programming languages use data dependency to determine the order in which calculations are performed. If Block B uses the output of Block A, then Block B is said to be dependent on Block A. Dependency propagates through blocks, so if C uses B's output then C depends on A (see Figure 3).

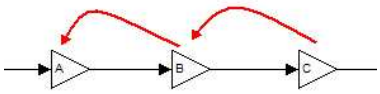

Figure 3. Data dependencies.

B(n) = f(A(n))
C(n)= f(B(n))
C(n) = f(f(A(n)))

Blocks that maintain state information, such as integrators and unit delays, break data dependency for the blocks following them (see Figure 4).

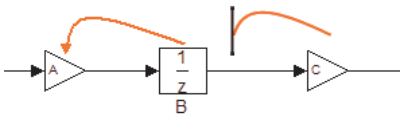

Figure 4. Effect of unit delays.

B(n) = f(A(n))
C(n) = f(B(n-1))

In Simulink the same principle can be extended to atomic subsystems.

UNDERSTANDING DATA DEPENDENCIES IN GRAPHICAL SYSTEMS - Complications arise when feedback loops are introduced into the system. Again, using a simple block example, we can show how feedback loops introduce ambiguous execution order into the system (see Figure 5).

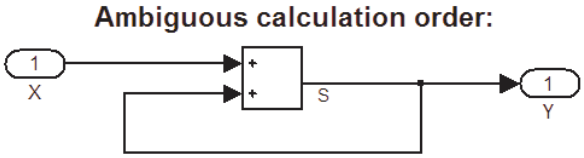

Figure 5. Ambiguous calculation order.

Note: The subscript letter "i" indicates the iteration count.

$Y_i = S_i$
$S_i = X_i + S_i$

Subtracting S_i from both sides (see Figure 6) results in:

$X_i = 0$

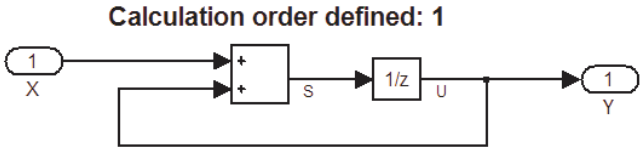

Figure 6. Data dependencies resolved, example 1.

$Y_i = U_i$
$U_i = S_{i-1}$
$S_i = X_i + U_i$

By substitution:

$Y_i = X_{i-1} + Y_{i-1}$

The resulting equation is not ambiguous; however, the output result is based on a delayed value of the input X (see Figure 7).

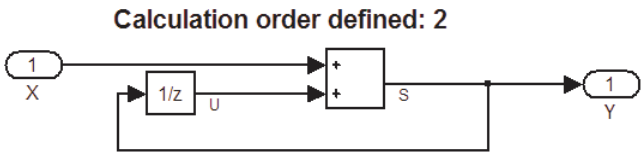

Figure 7. Data dependencies resolved, example 2.

$Y_i = S_i$
$S_i = X_i + U_i$
$U_i = S_{i-1}$

By substitution:

$$Y_i = X_i + S_{i-1} = X_i + Y_{i-1}$$

This configuration of the block diagram results in the desired calculation; the output value is based on the current value of X and the last value of Y.

METHOD FOR CONTROLLING CALCULATION ORDER

The method for resolving execution order is dependent on use of the Simulink loop diagnostic error messages. When Simulink finds a data dependency in a model, it returns an error message (see Figure 8).

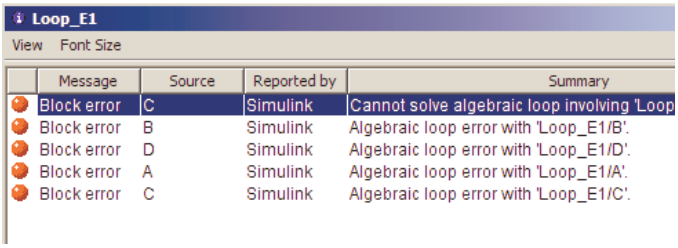

Figure 8. Example error message (direct backward).

The error messages have two forms: *direct backward loop* and *multiple coexisting loop* (see Table 1). The distinguishing characteristic of a direct backward loop is that when you click on the error messages in order, the signal flow between subsystems is direct and constant. In multiple coexisting loops, the order of the error message is "broken." At some point in the error message, either the block order jumps without a direct connection between the subsystems or the connecting signals are broken by a unit delay. Both types of error message can be used to resolve model data dependencies.

	Direct Backward	**Multiple Coexisting**
Number of loops in system	1	2 or more
Can be directly traced from error message	Yes	No; error message "jumps" at location of extra loops
Can be broken at a single point between two subsystems	Yes	No
Requires multiple iterations to resolve loops	No	Yes

Table 1. Loop types.

THE CONNECTION TABLE - The first step in resolving the data dependencies is building a connection table. The connection table is an aid to understanding where loops exist.

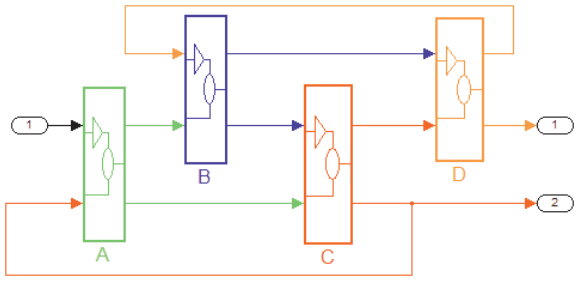

Figure 9. Example model.

Note: In this example model (Figure 9. Example model.), the subsystems are colored to make tracing signals easier. The colors do not represent the sample time for the model.

	C	B	D	A	Length
C	-	In	Out	In/Out	0
B	Out	-	In/Out	In	0
D	In	In/Out	-	X	0
A	In/Out	Out	X	-	0

Table 2. Connection table.

The connection table is built from the Simulink error message (Figure 8) for the example model (Figure 9). The information is collected based on the following rules:

- The column header is the order the subsystems appear. The row header is based on their appearance in the error message.
  - **In:** The row subsystem receives an Input from the column subsystem
  - **Out:** The row subsystem has an Output that goes to the column subsystem
- If the subsystem has both an In and Out, the In is listed first.
- The Length column is the number of columns between an Input and Output.

USING THE CONNECTION TABLE - The connection table is used to determine where the loops can be broken. The heuristic rule for solving the loop is:

- Starting at the top of the table, break the shortest length loops (e.g., all 0 length loops).
- Break the loop by inserting a unit delay at the input to the row subsystem.
  - If there are multiple inputs, place unit delays on all of the inputs.
- When all loops of a given length are broken, run the update diagram function to regenerate the table.

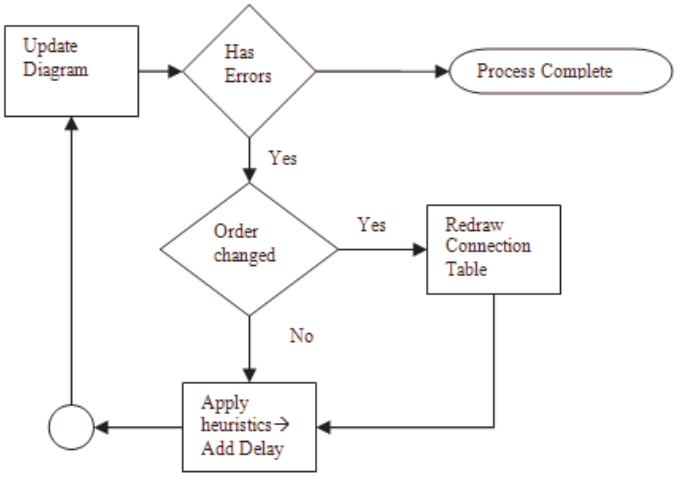

Figure 10. Heuristic flow chart.

In this example, there is a set of 0 length loops (A/C and B/D). Following the heuristic, we insert a unit delay at the input to A and B.

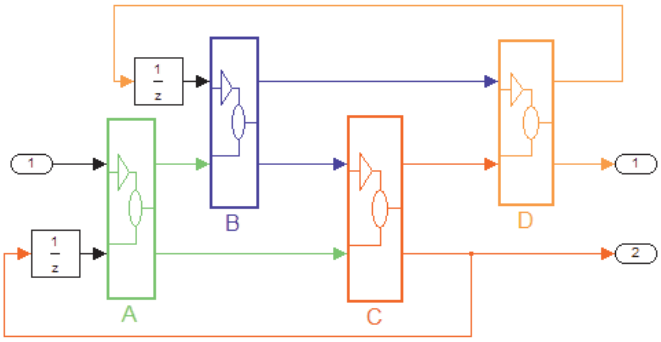

Figure 11. Data dependencies resolved, example 3.

It is a common mistake when dealing with systems such as this to add the unit delay before the import to subsystem D (B/D loop). Although doing so breaks the direct link between B and D, there is still an indirect loop (B C D B). The Simulink error diagnostic sorts the priority of the loops and allows for the minimum number of unit delays to be added.

A second example with a more complex model is provided in the appendix.

MULTIRATE SYSTEMS - There are two cases for multirate systems: a multirate system with single-rate components, and a multirate system with multirate components. The first case, a multirate system with single-rate components, is treated like the single-rate example already provided. When breaking the loops between multirate systems, a Rate Transition block is used instead of the Unit Delay block.

The multirate system, multirate component case is more difficult. The initial error diagnostic will not provide enough information to correctly reduce the problem to a set of nodes and loops. The method for resolving this issue is to first insert Rate Transition blocks between all signals running at different rates. Once this has been done, the method outlined earlier in the paper can be applied.

CONCLUSION

This paper has presented a simple heuristic for resolving data dependency loops within graphical models. As the examples demonstrate, the technique can be applied to both small and large models.

Future work will look at extending this methodology to an automated tool that would resolve data conflicts. Human input would still be used when multiple valid methods of resolving the problem exist.

REFERENCES

1. Guasch, A. et al, “Precompiled Submodels: A General Sorting Procedure,” Proceedings of the 2nd European Simulation Congress, Sep. 9–12, 1986, pp. 172–178.
2. Thomas, Drea, “Q&A,” The MathWorks MATLAB Digest, vol. 3(1):1–8 (1995).
3. Mosterman, Pieter J. et al, “Using Interleaved Execution to Resolve Cyclic Dependencies in Time-Based Block Diagrams,” 43rd IEEE Conference on Decision and Control, Dec. 14–17, 2004, pp. 4057–4062.
4. Pearce, J.G., “The Submodel Concept in Continuous System Simulation Languages,” Simulation of Dynamical Systems, IEEE Colloquium on Simulation of Dynamical Systems, pp. 2/1–2/3 (2002).

APPENDIX

Figure 12 shows a larger, more complex model with multiple coexisting loops. For this model, the process of removing the loops is more complex. In the case presented earlier in this paper, the error message was easily interpreted because there was only one loop in the system. When there is more than one loop, the error message may be less clear. However, the same methods can be used to determine useful information.

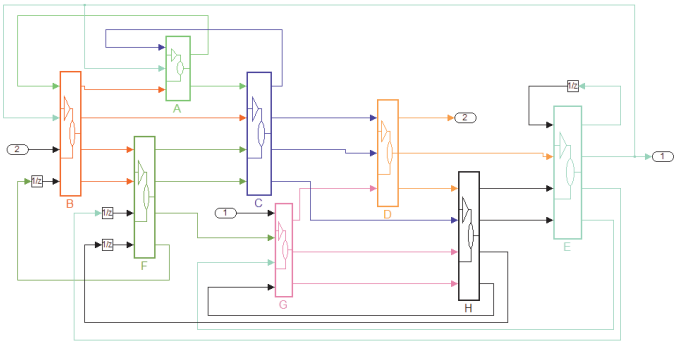

Figure 12. Complex example model.

	Message	Source
●	Block error	H
●	Block error	D
●	Block error	C
●	Block error	A
●	Block error	E
●	Block error	B
●	Block error	F
●	Block error	G
●	Block error	H

Figure 13. Example error message.

The gray shaded boxes in the updated connection table (see Table 3) represent subsystem pairs that are directly part of the error message loop. The table has four subsystems that have both input and output connections: A/C, B/A, G/H, and F/B. The subsystem pair F/B can be ignored because the output connection is already broken by a unit delay (denoted by the * symbol).

	H	D	C	A	E	B	F	G
H	NA	In	Out	X	Out	X	Out*	In/Out
D	Out	NA	In	X	Out	X	X	In
C	Out	Out	NA	In/Out	X	In	In	X
A	X	X	In/Out	NA	In	In/Out	X	X
E	In	In	X	Out	NA	X	Out*	Out
B	X	X	Out	In/Out	In	NA	In/Out*	X
F	In*	X	Out	X	In*	In/Out	NA	Out
G	In/Out	Out	X	X	In	X	In	NA

Table 3. Connection table.

- Shortest loop: G/H
  - Inserted break on Inport to G.
- Update diagram: No change to error message.
- Shortest loop: B/A
  - Inserted break into B because two out of three inputs are already broken.
- Update diagram: No change to error message.
- Shortest loop: A/C
  - Inserted break before Inport for C.
- Update diagram: Error message is updated.
- Create new connection table.

Figure 14 and Table 4 show the resulting model and connections.

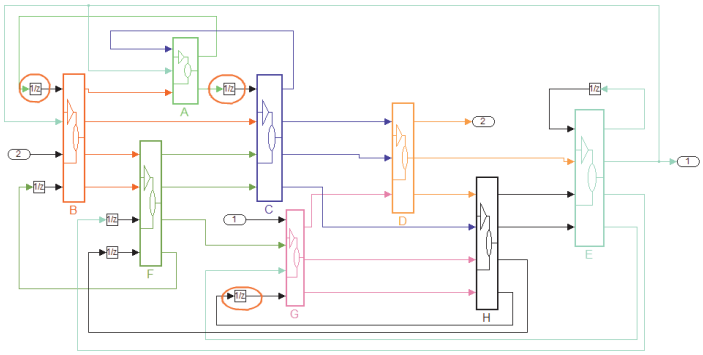

Figure 14. Example error message.

	H	D	C	B	E	F	G
H	-	In	In	X	Out	Out*	Out/In*
D	Out	-	In	X	Out	X	In
C	Out	Out	-	In	X	In	X
B	X	X	Out	-	In	Out	X
E	In	Out	X	Out	-	Out*	Out
F	In*	X	Out	Out/In	In*	-	Out
G	Out/In*	Out	X	X	In	In	-

Table 4. Updated connection table.

The break in the connection table occurs at F/E. The error message shows a loop even though the loop is broken by a unit delay. Because of this we start by working on entries in rows H~E, prioritizing the bottom of the table.

- Shortest loop: E and G (change from Output at E/G to Input at E/H)
  - Inserted break on the input to G.
- Update diagram: No change to error message.
- Shortest loop: B and E (change from Output B/C to Input B/E)
  - Inserted break on the input to B.
- Update diagram: No error messages.

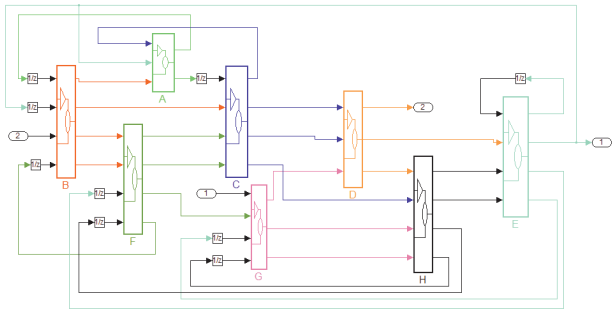

Figure 15. Example error message.

The final diagram, Figure 15, not all of the feedback loops are explicitly broken. For example, the feedback C to A and E to A are not broken. A common mistake would have been to insert the loop breaker at the Inports to A. This would not have resolved the data dependencies and would have required additional (redundant) unit delays.