OPTIMISATION OF MODEL EXECUTION TIME IN THE DEVSIM++ ENVIRONMENT

Yeong Geol Kim, Myung Soo Ahn and Tag Gon Kim

Dept. of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1, Kusong-Dong, Yusong-Gu, Taejon, 305-701, KOREA
ygkim@coregate.kaist.ac.kr
tkim@ee.kaist.ac.kr

KEYWORDS

DEVS Formalism, DEVSim++, Model Structuring, Simulation Speedup

ABSTRACT

The DEVSim++ environment is a C++ based realization of the DEVS formalism, which supports specification of discrete event systems in a modular, hierarchical form. This paper describes several approaches toward optimizing model execution time in the DEVSim++ environment. Such approaches include data structure optimisation and model structure optimisation. Experimental results for simulation speedup with the approaches will be shown.

1 Introduction

The DEVS (Discrete Event Systems Specification) formalism, developed by Zeigler ([Zei84]), supports specification of discrete event models in modular, hierarchical manner. Several realizations of the formalism are in place, the first of which was introduced in [Kim90]. DEVSim++ is an object-oriented environment for specifying discrete event models within DEVS formalism and executing such DEVS models in C++. Since the first version of the DEVSim++ environment is developed in 1992, we have put much effort to reduce simulation execution time within the environment.

Such effort ranges from data structure optimisation to model structure optimisation. This paper describes an approach toward reducing simulation time in the DEVSim++ environment. It also shows experimental results for simulation speedup using various DEVSim++ models.

In Section II, the DEVSim++ environment is breifly introduced with overview of its approach for reducing simulation speedup. Section III and IV presents data structure optimisation and model structure optimisation approach, respectively. Section V shows experimental results with various DEVS models. Section VI concludes our discussion.

2 Optimisation Approach in the DEVSim++ Environment

DEVSim++ is an object-oriented environment for discrete event simulation, which implements DEVS formalism in C++ ([KP92]).

The environment supports modelers to develop discrete event models using the hierarchical composition technology within the DEVS framework. For simulation, DEVSim++ implements hierarchical scheduling algorithms in abstract simulators of atomic and coupled models. Details of the environment will be found in ([Ahn94]).

2.1 Simulation Algorithm in DEVSim++

Figure 1 shows a hierarchical DEVS model and its tree representation. The outmost model M123 is at the root node of the tree, and its components M1 and M23 are represented as children nodes of root, and so on. For each model in the tree, associated is an abstract simulator which implements the hierarchical simulation algorithm proposed by the DEVS theory. During simulation, model components are passive ones

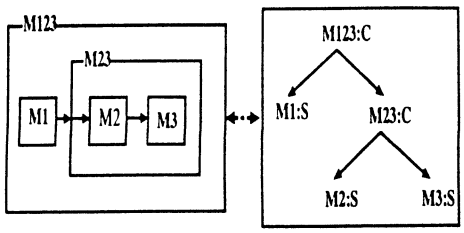

Figure 1: Model Hierarchy & Tree Representation

which are operated by active associated simulators. That is, a DEVS model is activated only when the associated simulator requests. The DEVSim++ environment implements two types of simulators proposed in ([Zei84]) as classes: *Coordinator* associated with each coupled model and *Simulator* assocaited with each atomic model.

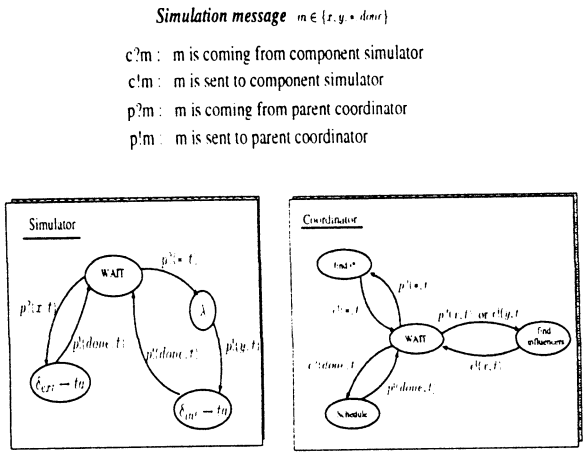

Figure 2: Abstract Simulators in DEVSim++

There are four kinds of simulation messages as shown in Figure 2, namely x, y, $*$, *done* message, each of which is associated with current simulation time t. Simulation proceeds by means of changing such messages between abstract simulators.

Simulator receives two kinds of messages, (x,t) and $(*,t)$. When a *Simulator* receives a (x,t) message from its parent *Coordinator*, it executes δ_{ext} and updates a new schedule with time advance function *ta*. If it receives a $(*,t)$ message, it executes output function λ to prodcue output message (y,t), and then δ_{int} followed by *ta*. After executing *ta* in both cases, the *Simulator* should send a $(done, tN)$ message to its parent coordinator to notify a new schedule tN.

On the other hand, *Coordinator* receives all kinds of simulation messages. If it receives (x,t) or (y,t) messages, it routes the message to a destined component simulator(s). And when receiveing a $(*,t)$ messange, the coordinator finds an imminent componet(s) with the minimum schedule time to route the $(*,t)$ message. Finally, receiving a $(done, tN)$ message from component simulators selects the minimum schedule from those of components to send this information to its parent *Coordinator*.

2.2 Optimisation Approaches

Due to the underlying formalism, models developed in DEVSim++ is in hierarchical modular form. Such form has advantages in the model verification phase, which is essentially programs debugging. However, a hierarchical model so developed may take much simulation time than a behaviorally equivalent model with different structure in the data collection phase. To take advantages from both phases, a verified hierarchical model is transformed into an optimized model structure for execution time. The approach for optimisation combined the profile anlaysis of various simulation run with simulation time estimation from the model structure. The followings are two optimisation techniques:

- Profile based optimisation : Data Structure optimisation
  - DEVSim++ V1.1 to DEVSim++ V1.3
- Simulation time estimation based optimisation : Model Structure optimisation
  - DEVSim++ V2.0

3 Data Structure Optimisation

This section describes data structure optimisation for simulation speedup from DEVSim++ V1.1 to DEVSim++ V1.3.

3.1 Integration of Model & Simulator Classes : DEVSim++ V1.1

DEVSim++ V1.0 separates class definitions of models and associated simulators to reuse models

through composition technology. To remove an overhead from message exchanges between a model and an associated simulator, the pair of model-simulator classes was simply integrated into a single class in DEVSim++ V1.1.

3.2 Management of schedule : DEVSim++ V1.2

DEVSim++ V1.0 used linked list data structure for schedule time management, and it managed all schedules including *infinity* time schedule [1]. To reduce an overhead from the linked list, DEVSim++ V1.2 employed *Min Heap* data structure for schedule time management.

3.3 State variables : DEVSim++ V1.3

Every atomic model has its own state variables to represent its states. Whenever state transition occurs in an atomic model, it should find an appropriate state variable to modify. DEVSim++ V1.0 employed the type string to represent a state variable name, which inevitably caused a lot of string comparisons. DEVSim++ V1.3 replaces the string type by the enumeration type for state variables to reduce the string comparison overhead. The result is that any state variable can be found directly by an array index.

Model Name	# of atomic models	Execution time (secs.) / Speedup (w.r.t DEVSim++ v1.0)			
		DEVSim++ V1.0	DEVSim++ V1.1	DEVSim++ V1.2	DEVSim++ V1.3
CSMA_CD	18	3180.9 / 1.00	2765.2 / 1.15	2470.3 / 1.29	2191.2 / 1.46
HyperCube	512	3480.1 / 1.00	3332.5 / 1.04	3135.1 / 1.11	2310.9 / 1.51
Columbian health care	14	3205.8 / 1.00	2829.2 / 1.13	2959.2 / 1.08	618.9 / 5.18
FTOS	93	2662.3 / 1.00	2590.8 / 1.11	2664.2 / 1.08	2307.7 / 1.24
Token Ring	80	3209.2 / 1.00	2560.1 / 1.25	2650.1 / 1.21	1428.7 / 2.25
				Average speedup	**2.23**

Figure 3: Experimental results of speedup for DEVSim++ versions

Figure 3 shows experimental results with five DEVS models, developed at KAIST CORE LAB, which have been used as a benchmark to measure simulation speedup for various DEVSim++ versions. Note that the DEVSim++ version 1.3 has achieved speedup of 2.23 against the DEVSim++ version 1.0.

[1] infinity time schedule means that a model will never be activated on its own without external event

4 Model Structure Optimisation

This section describes the DEVSim++ V2.0 environment in which optimal model structure is obtained such that expected simulation time is minimized.

The notations frequently used are summarized.

Notation	Meaning
N	# of atomic models
Ai	total # of input ports of atomic models
Ao	total # of output ports of atomic models
Ci	total # of input ports of coupled models
Co	total # of output ports of coupled models
k_i	avg. # of input ports for a coupled model
k_o	avg. # of output ports for a coupled model
Mx	frequency of x message occurences
My	frequency of y message occurences
$M*$	frequency of $*$ message occurences
Md	frequency of d message occurences
Tx	function execution time for a x message
Ty	function execution time for a y message
$T*$	function execution time for a $*$ message
Td	function execution time for a d message
S_{rm}	frequency of schedule removal
S_s	frequency of schedule search

Table 1: Notations

In this environment, simulation time is estimated from model structure information.

- N, Ai, Ao
- Coupling topology between atomic models

We estimate total simulation overhead T_{ov} as:

$$T_{ov} = M_{ov} + S_{ov} \quad (1)$$

M_{ov} represents a message handling overhead between hierarchical models, whereas S_{ov} represents a scheduling overhead due to the hierarchical scheduling algorithm.

$$M_{ov} = M_x \times T_x + M_y \times T_y + M_* \times T_* + M_d \times T_d \quad (2)$$

$$S_{ov} = S_{rm} \times T_{rm} + S_s \times T_s \quad (3)$$

Futhermore, S_{rm} and S_s are related to occurences of simulation messages in DEVSim++ as follows.

$$S_{rm} = p_a \times (M_x + M_y + M_*) + M'_d \quad (4)$$

$$S_s = p_s \times M_d \quad (5)$$

where, M'_d is equal to number of star messages sent down from the outmost model, and p_a is probability that an atomic model is active in schedule time during simulation.

$$M'_d = A_o \quad (6)$$

p_s is probability that all components in the waiting-list of a coupled model have finished their state transition so that new minimum schedule should be updated to it.

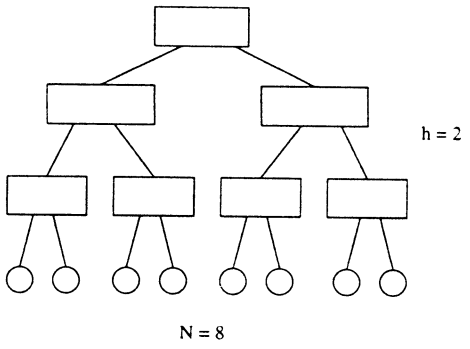

Figure 4: Model structure with parameter h

For simplicity, we restrict the tranformed model structure such that all coupled models have only two children except parents of atomic models. With this restriction, T_{ov} becomes function of the tree height h. Figure 4 shows one feasible model structure with $h = 2$ and $N = 8$. Then, we can estimate M_x, M_y, M_*, M_d in terms of h with following assumptions.

- M_x is proportional to total number of input ports of coupled models
  - $p_o \times C_i$
- M_y is proportional to total number of output ports of all models
  - $p_o \times (C_o + A_o)$
- M_* is proportional to total number of descendant atomic models' output ports for all coupled models
  - $M_* = A_o * h$
- M_d is sum of x and $*$ messages sent to all models
  - $M_d = M_x + M_* + p_o \cdot A_i + A_o$

p_o is probability that an atomic model will produce an output for each active state.

From the above equations, the probabilistic parameters p_o, p_a and p_s play an important role in accurate estimation of simulaton overhead. Since the parameters mainly depend on model behavior rather than model structure, sample simulation is employed for the estimation. It is observed from experiments that less than one minute sample simulation is sufficient for the values of parameters to converge. Note that the sample simulation process is automatically performed within DEVSim++ V2.0.

Therefore, the estimated values of M_x, M_y, M_*, and M_d are as follows :

$$M_x = p_o \cdot C_i = 2 \cdot p_o \cdot k_i \cdot (2^h - 1) \quad (7)$$

$$M_y = p_o \cdot (C_o + A_o) = p_o \cdot \{2 \cdot k_o \cdot (2^h - 1) + A_o\} \quad (8)$$

$$M_* = A_o \cdot h \quad (9)$$

$$M_d = 2 \cdot p_o \cdot k \cdot (2^h - 1) + A_o \cdot h + p_o \cdot A_i + A_o \quad (10)$$

Similary, S_{rm} and S_s are obtained.

Note that T_x, T_y, T_*, T_d are constant, while T_{rm} and T_s are linearly dependent on the average number of components, say Q

$$Q = p_a \cdot \{\frac{2h}{h + 1 + p_o} + \frac{(1 + p_o) \cdot N}{(h + 1 + p_o) \cdot 2^h}\} \quad (11)$$

$$T_{rm} = Q \times t_{rm} \quad (12)$$

$$T_s = Q \times t_s \quad (13)$$

As a result, T_{ov} is obtained as follows.

$$T_{ov}(h) = c_1 \cdot 2^h + c_2 \cdot h + c_3 \cdot 2^{-h} + c_4 \cdot h \cdot 2^{-h} + c_5$$

The coefficients c_1 to c_5 are easily obtained from equation (1) to (13).

To find optimal parameter h^* which minimizes $T_{ov}(h)$, we calculate 1st derivative of $T_{ov}(h)$ with respect to h, and equate it with zero ($0 \leq h < log_2 N$).

- $T'_{ov}(h) = 0$

Then, it is reduced to following equation.

$$c_1 \cdot ln_2 \cdot 2^h + (c_3 - c_4 \cdot ln_2 - c_3 \cdot ln_2 \cdot h)2^{-h} + c_2 = 0 \quad (14)$$

The above equation can be easily solved by well-known numerical methods such as the Newton method.

5 Experimental Results

N	100	400	1600	3200
h^*_{expr}	0	2	4	4-5
h^*_{theory}	**0**	**2**	**4**	**5**
SPEEDUP	1.29	1.29	1.77	1.73

Table 2: CSMA_CD: expectation of h^* and speedup

N	512	1024	2048	4096
h^*_{expr}	1	1-2	3	3-4
h^*_{theory}	**1**	**2**	**3**	**4**
SPEEDUP	2.04	2.13	1.23	2.01

Table 3: HCUBE: expectation of h^* and speedup

N	80	320	1280	3840
h^*_{expr}	2	3-4	4-6	5-8
h^*_{theory}	**2**	**4**	**6**	**8**
SPEEDUP	1.23	3.31	11.6	18.4

Table 4: Token Ring: expectation of h^* and speedup

Tables 2 to 4 show comparisons of optimal tree hights obtained experimentally (h^*_{expr}) against those obtained theoretically (h^*_{theory}). It also shows speedup of the optimal model structures against original model structures. Note in the tables that "3-4" means that simulation time with the 3-level structure was almost equal to that with the 4-level structure. As the tables show, the analytic method of finding the optimal model structure is well suited for minimizing simulation run time.

6 Conclusion

This paper has shown a methodology for reducing simulation time for hierarchical DEVS models developed in the DEVSim++ environment. The methodology has employed a static optimisation approach based on profile analysis, which consists of data structure optimisation and model structure optimisation. Experimental results have proved that the methodology markedly reduced simulation time for various hiearchical models developed in the DEVSim++ environment.

References

[Ahn93] Myung S. Ahn and Tag G. Kim, "DEVS Methodology for Evaluating Time-Constrained Message Routing Policies," in *Discrete Event Dynamic Systems : Theory and Applications*, 1993.

[Ahn94] Myung S. Ahn and Tag G. Kim, *DEVSim++ User's Manual*, available by anonymous ftp to sim.kaist.ac.kr, 1994.

[Kim90] T. G. Kim and B. P. Zeigler, "The DEVS-Scheme Simulation and Modelling Environment," Chapter 2 in *Knowledge Based Simulation: Methodology and Application* (eds:Paul A. Fishwick and Richard B. Modjeski) Springer Verlag., Inc, pp.20-35, 1990.

[KP92] T. G. Kim and S. B. Park, "The DEVS formalism: Hierarchical modular systems specification in C++," in *Proceedings of European Simulation Multiconference*, 1992.

[Zei84] B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation.* Academic Press, 1984.