DEVS Execution Acceleration with Machine Learning

Hesham Saadawi
Government of Canada
Ottawa, ON
hsaadawi@hotmail.com

Gabriel Wainer
Carleton University
Ottawa, ON
Gabriel.Wainer@sce.carleton.ca

ABSTRACT
Discrete Event System Specification DEVS separates modeling and simulation execution. Simulation execution is done within a runtime environment that is often called a DEVS simulator. This separation creates an opportunity to incorporate new smart algorithms in the simulator to improve simulation execution. We propose incorporating predictive machine learning algorithms into the DEVS simulator in order to cut simulation execution times significantly for many simulation applications without compromising the simulation accuracy. We introduce a specific learning mechanism that can be embedded into the DEVS simulator to incrementally build a predictive model that learns from past simulations. We further look into issues related to the predictive model selection, its prediction accuracy, its effect on the overall simulation performance, and when to switch between the predictive model and the simulation during an execution.

Author Keywords
Discrete Event System Specification DEVS; Machine Learning; Regression Models.

ACM Classification Keywords
I.6.1 SIMULATION AND MODELING (e.g. Model Development).

1. INTRODUCTION
Computer Simulation has become an important tool for the advancement of many disciplines in science and engineering. As the complexity of these systems grows, the scale of the computer simulations that represent them also grows.

High-performance computing HPC platforms that exploit the parallel processing of hundreds or thousands of computing nodes are becoming increasingly used to execute large-scale simulations that otherwise would take prohibitively-huge amount of time to execute on single processing machinery. However, HPC platforms are expensive and need advanced simulation code can execute in parallel fashion while keeping any synchronization issues under control. Reducing the computing power needed for large-scale simulations would be an advantage, as this would reduce both the computing cost and simulation execution time, hence allowing for accelerated design cycles.

Traditional simulation code is usually written with procedural programming languages such as FORTRAN, C, C++, etc. A great effort is needed to optimize this code to execute faster, and usually these optimizations done on one platform do not necessarily work on another platform. This makes code optimizations or other techniques to accelerate simulation execution, as we will discuss in this paper, limited to a particular model and simulation code and cannot be applied easily to other models or simulation platforms.

Discrete Event System Specification DEVS on the other hand, separates the model from its execution. The execution is done by a *DEVS simulator* which is a runtime environment that executes all types of DEVS models, and can execute on different platforms. This creates an opportunity to implement simulation optimization techniques into this common-for-all runtime environment. Future scientists and engineers would focus only on the modeling task at hand and leave simulation execution optimization to the DEVS simulator. These optimizations would execute on different platforms as the need comes to scale the simulation, without affecting the model structure of behavior.

In this paper, we introduce a technique to optimize DEVS simulation using regression modeling from machine learning (ML). With this technique, a predictive model is built incrementally using information from past simulation executions. This model would approximate the behavior of complex components of a DEVS model, while executing with much less computing resources. As a typical large-scale DEVS model executes, many components have repetitive computations represented as the same inputs, while in same state. A great number of computations can be saved if the component behavior is learned from past experiences and then output is predicted instead of computed.

2. RELATED WORK
Recently, some researchers have used techniques of machine learning to accelerate the simulation in particular scientific and engineering domains. For example, in CPU instruction set simulators [1], a machine learning algorithm was used to build a regression model. This model was used during simulation to predict coarse-grained simulation results, thus saving computational time. However, this technique, intuitively, consumed computation resources during the learning phase and thus showed to be better used for long-running simulations where this consumption can be offset by savings in the simulation by using the built predictive model. The authors report moderate simulator speedups of about

50% for this type of application, and with an error margin of that is mostly below 5%. The authors implemented an algorithm that decides when to switch between detailed simulation and when to use the predictive model.

In the domain of materials science, some researchers used machine-learning algorithms to incrementally build a predictive model to reduce complex Quantum mechanics-based ab initio molecular dynamics (MD) calculations during the simulation phase [2]. Once enough training data is obtained from the simulation, an algorithm can decide if using the predictive model is accurately sufficient or it needs to run the simulation. This saves the need to do many complex and repetitive calculations during the simulation. Once the machine learning model is built from the training set, the authors note that a prediction using this model is faster than MD calculations by an order of 10^6.

3. BACKGROUND

3.1. DEVS

We define here the classical DEVS [5]. First we define DEVS atomic model. The DEVS Atomic Model is defined as:

$AM_{TC} = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$

- X : The set of external inputs.
- Y : Set of external outputs.
- S: set of system states.
- δ_{int}: $S \rightarrow S$ is the internal transition function.

– δ_{ext}: $T \times X \rightarrow S$ with $T=\{(s,e) |$ s $0 \le e \le ta(s),\ e \in \mathbb{R}_{0,+\infty}\}$ is the external transition function (e is the time elapsed since the last transition, which takes a positive real value).

- λ: $S \rightarrow Y \cup \varnothing$ is the output function.
- ta: $S \rightarrow \mathbb{R}_{0,+\infty}$ is the time advance function that maps each state to a real number.

Coupled DEVS models are composed of atomic or other coupled DEVS models:

$$CM \equiv \left\langle X, Y, D, \{M_i\}, C_x, C_y, Select \right\rangle$$

X: Set of external input events.
Y: Set of external output events.

D: Finite index of sub-components.

$\{M_i\}$: The set of sub-components. A sub-component may be an atomic or coupled. $i \in D$ is the index of the component.

C_x: Set of input couplings.
C_y: Set of output couplings.

Select: $2^D \rightarrow D$ is a tie-breaking function, which defines how to select an event from asset of simultaneous events.

A coupled DEVS model M can be simulated with an equivalent atomic DEVS model, whose behavior is defined as follows [3]:

$$M = < X, Y, S, s_0, \delta_{ext}, \delta_{int}, \lambda, ta >$$

- X and Y are the input and output event sets, respectively. X is the set of all input events accepted and Y is the set of all output events generated by coupled model M.
- $S = \times_{i \in D} V_i$ is the model state. It is expressed as the Cartesian product of all component states, where V_i is the total state for component i, $V_i = \{(s_i, t_{ei}) \mid s_i \in S_i, t_{ei} \in [0, ta(s_i)]\}$. Here, t_{ei} denotes the elapsed time in state s_i of component *i,* and S_i is the set of states of component *i*.
- $s_0 = \times_{i \in D} v_{0i}$ is the initial system state, with $v_{0i} = (s_{0i}, 0)$ is the initial state of component $i \in D$.
- $ta : S \rightarrow \mathrm{T}$ is the time advance function. It is calculated for the global state $s \in S$ of the coupled model as the minimum time remaining for any state among all components, formally: $ta(s) = \min\{(ta(s_i) - t_{ei}) \mid i \in D\}$ where $s = (\ldots, (s_i, t_{ei}), \ldots)$ is the global total state of coupled model at some point in time, s_i is the state of component *i*, t_{ei} is elapsed time in that state.
- $\delta_{ext} : X \times V \rightarrow S$ is the external transition function for the coupled model. Where V is total state of the coupled model: $V = \{(s, t_e) \mid s \in S, t_e \in [0, ta(s)]\}$.
- $\delta_{\mathrm{int}} : S \rightarrow S$ is the internal transition function of the coupled model.

$\lambda : S \rightarrow Y$ is the output function of the coupled model.

3.2. MACHINE LEARNING

Machine learning techniques and algorithms goal is to identify patterns from data. These techniques are usually divided into two categories, supervised and unsupervised learning.

In supervised learning, the data is split into two or more sets, a training set and a test set. The algorithm builds a model using the training set, and then tests the accuracy of it against the test set. If accuracy found to be within acceptable limits, the model would then be used to predict new output values from new inputs. Usually these models are best used as interpolative instead of extrapolative, i.e. predict data points that lie in the range of training set data to have reasonable accuracy in the predicted output.

In unsupervised techniques, the algorithm deals with all the data and tries to identify groupings in the data to help in the task of data classification.

For our purpose in this paper, we are interested with the supervised learning algorithms as the DEVS simulator execution would provide us with a *training* and a *test* sets that are known to be correct.

4. EXPRESSING DEVS COUPLED MODELS AS A PREDICTIVE MODEL

It is a well-known fact as shown in [5] that each DEVS coupled model has an equivalent atomic model behavior. This behavior can be deterministic or non-deterministic depending on the functions defining the behavior of the model. We consider here only deterministic DEVS models. Further, we limit the discussion below on DEVS models that represent continuous systems as described in [6][7]. In this case, the behavior of a coupled or atomic DEVS model can be approximated with a predictive model. In other words, the equivalent transition functions are approximated with a predictive model. Let's look at the *observed* behavior of a DEVS model when receiving an external input X and producing an output Y after some time t:

$$\delta_{ext}(S_1,e,X) = (S_2,\sigma)$$

Where S_1 is the current model state, e is the elapsed time in that state, $\sigma = ta(S_2)$ the time advance value of state S_2.

$$\lambda(S_2) = Y$$

$$\delta_{int}(S_2) = (S_3)$$

Combining these functions produces:

$$Y = f_1(S_1,e,X) \qquad \text{Eq. 1}$$

$$S_3 = f_2(S_1,e,X) \qquad \text{Eq. 2}$$

$$\sigma = f_3(S_1,e,X) \qquad \text{Eq. 3}$$

f_1 and f_3 express the coupled model behavior as a relationship between the model input X, its state (S_i,e) and observed output Y that is observed at a time instant of σ *EXPLAIN BETTER* . These two functions are equivalent to δ_{ext} and λ. f_2 represents the model internal transition function.

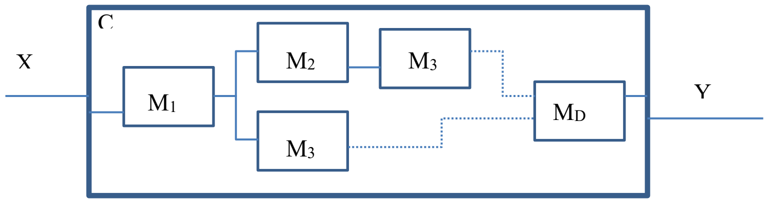

Figure 1: DEVS Coupled Model

Figure 1 shows a general DEVS coupled model C that composes of D number of DEVS sub models (atomic or coupled) and has an input vector X and produces and output vector Y. The total state of this model can be represented as a vector of the states of all components

$S = (s_1,s_1,\ldots.s_l)$

Each of subcomponent state s_i can be generalized to have a vector of real variables $s_i = (v_{i1},v_{i2},\ldots)$ describing the subcomponent state. Therefore, the total state of model C at any instance in time can be expressed as a vector of all real variables representing subcomponent states

$S = (s_1,\ldots.s_n)$, where $n = \sum_{i=1}^{i=D} |s_i|$

Therefore, the model state can be generalized as a vector of n real numberS giving $S \in \mathbb{R}^n$. External inputs can also be generalized as a vector of one or more real numbers $X \in \mathbb{R}^k$. External outputs can also be generalized as vector of one or more of real numbers $Y \in \mathbb{R}^m$. Formally,

$$S = (v_1, v_2, v_3, \ldots, v_n),\ \ n > 0\ ,\ v_i \in \mathbb{R},\ 1 \le i \le n$$

$$X = (x_1,x_2,\ldots,x_k),\ k > 0,\ ,\ x_i \in \mathbb{R},\ 1 \le i \le k$$

$$Y = (y_1,y_2,\ldots,y_m),\ m > 0\ ,\ \ y_i \in \mathbb{R},\ 1 \le i \le m$$

We propose using machine learning techniques to build an approximate equivalent model that would help predicting the model outputs as defined by Eq. 1, Eq. 2, and Eq. 3. The advantage of doing this is shown in [1][2] that it can reduce simulation time significantly for complex coupled models. This is because of two reasons; first, a predictive model that consumes much less computations to produce an equivalent output can replace a computation-intensive DEVS model. Secondly, it is common in engineering and scientific models to use many instances of a component M_x to build a larger model. If this component M_x happens to consume relatively a lot of computing resources, it would be beneficial to replace it with an equivalent predictive model. Think for example about a transistor model in VLSI simulation. In [8], the authors used three-dimensional statistical numerical simulations to study a new design of a MOSFET transistor. This simulation is described as atomistic scale as it takes into account quantum behavior of electrons and holes. The resulting characteristic behavior of the transistor was obtained and plotted from the study. If this transistor were used in a VLSI, it would be a good candidate for building an equivalent predictive model for its characteristic curve instead of repeating expensive computations for each instance in a large VLSI model.

Further, once built, this predictive model is a characteristic behavior of the DEVS model. Therefore, it would be good to store it with the model for future simulations as long as the model definition does not change.

Doing so would help scale the DEVS simulation to handle system-of-systems as any subsystem behavior can be harvested during the simulation and then repetitive computations in that subsystem would be answered from the predictive model in a faster time and with fewer computations.

Choosing a predictive modeling technique to approximate f_1, f_2 and f_3 depends on several factors:

a. Efficient: as the model would be built during simulation time.

b. Incremental: it can accept new points added to the model with least computations.

c. First assumption is working with real numbers inputs and outputs ????.

d. Can work reasonably well with non-linear functions (Unknown functions that arise in solving physical ODE).

e. There is a way to give a ceiling of error for this model for new data

f. Can automatically correct/optimize for the least error by selecting higher orders of input variables. Assuming all input variables are contributing to the output.

Generally speaking, there are two predictive models of machine learning that can satisfy most of these factors, namely statistical regression, and artificial neural networks.

4.1. STATISTICAL REGRESSION MODELING

We introduce regression modeling, a simple method to build predictive continuous models from a set of known data [4]. Regression is used to predict the value of an output variable $y = f(x)$ based on the input of one or more *predictor* input variables $x = (x_1, . . . ,x_N)$. The function $f(x)$ can be a simple linear function or non-linear. If $f(x)$ is linear, the regression model is expressed as:

$$\hat{y} = \theta_1 + \sum_{i=2}^{i=N+1} \theta_i x_i + \varepsilon$$

Where $\hat{y}$ is the predicted value, $(\theta_0, \theta_1, \ldots, \theta_N)$ are model weights, and $\varepsilon = y-\hat{y}$ is the prediction error. This error is the difference between the actual y value and the estimated value $\hat{y}$ obtained from the model. The model is built from a training set of m data points $[(y_1,x_{11},x_{12},\ldots,x_{1N}),..,(y_m,x_{m1},x_{m2},\ldots,x_{mN})]$. Using least squares method, the weights of the model are calculated as follows in a matrix form

$$\Theta = (X^TX)^{-1}X^TY \qquad \text{Eq. 4}$$

Where $X=\begin{bmatrix} 1 & x_{11} & \cdots & x_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \cdots & x_{mN} \end{bmatrix}$, $\Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{bmatrix}$, $Y=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$,

X^T is the transpose of matrix X, x_{11} is the element of first data point in the training data set that correspond to the variable x_1, and the subscript -1 indicates the matrix inverse.

Once, we have the regression model, statistics estimates a variance of the predicted value from the actual value as

$$\varphi = \frac{\sum_{i=1}^{i=m}(y_i-\hat{y}_i)}{m-2} \qquad \text{Eq. 5}$$

In addition, the standard error of regression or the standard deviation would equal to$\sqrt{\varphi}$. With an assumption of errors to follow a normal distribution, an estimated value $\hat{y}$ is considered the mean of the distribution. Therefore, we can get an estimate of how far the actual y value from the predicted mean for a 95% confidence interval as

$$(\hat{y} - 1.96\sqrt{\varphi}) \geq y \geq (\hat{y} + 1.96\sqrt{\varphi}) \qquad \text{Eq. 6}$$

This model would approximate a continuous function with a linear one. However, if the approximated continuous function deviates significantly from a linear model, *f(x)* can take higher orders of x to build quadratic or cubic predictor functions that can better approximate the continuous function, i.e. average error is further minimized. Other non-linear forms of *f(x)* also exist for more complicated models. However, the model building would be more computationally demanding in this case.

The advantage of using regression models is that they are simple to construct and there are efficient incremental algorithms that can build the model incrementally as new data points arrive [1][9].

4.2. PROPOSED DEVS SIMULATOR PREDICTIVE MODELING

We propose an algorithm to be used in the simulator to aid in using predictive models in the simulation. Key ideas of this algorithm are:

a. Decide which model is a good candidate to be approximated with a predictive model: Apparently, models that are influenced and can influence a simulation are these with inputs and outputs. Further, models with significant internal computations and are involved in many model interactions are good candidates. We will elaborate on this criterion later in this paper.

b. Collect data for the chosen models: once a candidate model is identified, a data collection phase is begun. With enough data points, the predictive model is built. For regression models, usually the number of data points need to exceed the number of variables for the model construction to be solvable.

c. Decide when an accurate-enough predictive model is built and is ready for use: Eq. 5 shows that the variance of a predictive regression model can be decreased by including more m data points in the model construction. This in turn would reduce estimation error interval as defined in Eq. 6. It is also known for regression models that it is best to be used as interpolative, i.e. to predict new data points with inputs that lie in the range of the model training set.

d. Revert back to simulation execution if accuracy falls below accepted margin: this would happen if either not having enough data points, or an input comes to the model that is outside the range of inputs of the model training set. In this case, a new data point (input/state/output) should be added incrementally to the predictive model and its weights are updated accordingly. Another possible reason of reduced accuracy is if the chosen DEVS model behavior is nonlinear. In this case, a nonlinear predictive model either to be built with acceptable accuracy, or revert to the simulation for accurate results.

4.3. Estimating criterion for computational savings

As mentioned above in section 4.2, point a, the DEVS simulator would need a criterion to select a candidate DEVS model for building a predictive model. Ideally, this would save computations either in the current simulation run or future simulations where this DEVS component is taking part. Formally

$$\mathbf{C_{P.\,Model}} \leq \mathbf{C_{DEVS}} \qquad \text{Eq. 7}$$

Where:

$C_{P.\ Model}$ is the number of computations to build a predictive model.

C_{DEVS} : is the number of computations done by the DEVS model. As an approximation, this number is approximately the number of transitions done by a DEVS model to produce an output after receiving an external input.

Eq. 4 is used to build a predictive linear regression model. In that equation, the dominating term in determining the number of computations is the calculation of the matrix inverse. If we have an $m \times m$ matrix, then number of computation to calculate its inverse is approximately $2m^3/3$ operations [10]. Therefore, the above criterion can be expressed as

$$2m^3/\,3 \leq C_{DEVS}$$

The number of data points m to be observed and sampled during a simulation execution is based on the desired accuracy, which is expressed as a specific standard error for a desired confidence interval as in Eq. 6. This number would determine the expected number of computations to build a predictive model and thus can be a guide to select a DEVS model with comparable computations or more. Once a good predictive model for a DEVS component is built for a given accuracy, it is expected the predictive model would not change unless the DEVS component changes. Therefore, it would be beneficial to save this predictive model with the DEVS component definition to be used in future simulations to save even more computational resources. In this case, number of DEVS transitions in Eq. 7 would be total transitions during one or more simulation runs.

5. CONCLUSION AND FUTURE RESEARCH

In this paper, we proposed the use of machine learning techniques to build predictive models as replacements to computation-intensive DEVS components. Doing so, would accelerate the simulation of complex system-of-systems without sacrificing much accuracy. Some experimentation with this concept was implemented recently in the general simulation community, but its wide application to codes of simulations would be difficult without the concept of a *model* and a *simulator*.

The application of our proposed approach should be an integral part of the DEVS simulator as DEVS separates the model definition from its simulator. This would be a clear advantage when modeling scientific and engineering systems with DEVS.

We showed one method of machine learning, simple regression, that may be used to efficiently and incrementally build predictive models. Using this method, we showed a way to estimate the prediction error, which can be acceptable when confined to the range of error induced by the simulation approximations or quantization.

As the DEVS simulator does not know priori the expected behavior function of a component, or the number of state variables, it would be ideal to have other techniques available to build more complex predictive models. Regression models usually are not accurate enough if the function to approximate is non-linear above quadratic or cubic, or the number of variables is high. In this case, other ML techniques would be necessary. One of these techniques that are used to build complex non-linear predictive models is Artificial Neural Networks ANN. ANN can be trained with sample data, the training set, to build a predictive model that can be used to predict output value from the inputs. The drawback of this technique is its intensive computations and long training time. We plan to explore this option in future research.

In Future research, we also plan to implement these algorithms into a DEVS simulator and execute a benchmark system model to fine-tune our methods.

ACKNOWLEDGMENTS

…

REFERENCES

[1] Powell, Daniel Christopher, and Björn Franke. "Using continuous statistical machine learning to enable high-speed performance prediction in hybrid instruction-/cycle-accurate instruction set simulators." Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis. ACM, 2009.

[2] Botu, Venkatesh, and Rampi Ramprasad. "Adaptive machine learning framework to accelerate ab initio molecular dynamics." International Journal of Quantum Chemistry (2014).

[3] Wikipedia. DEVS behavior. http://en.wikipedia.org/wiki/Behavior_of_Coupled_DEVS. [Accessed: Dec. 2015].

[4] Weisberg, S. Applied Linear Regression, third edition, Hoboken NJ: Wiley (2005).

[5] BP Zeigler, H. Praehofer, T.G. Kim (2000) Theory of modeling and simulation, 2nd edn. Academic Press, New York.

[6] Francois E. Cellier and Ernesto Kofman (2006). Continuous System Simulation (first ed.). Springer. ISBN 978-0-387-26102-7

[7] James Nutaro (2010). Building Software for Simulation: Theory, Algorithms, and Applications in C++ (first ed.). Wiley. ISBN 0-470-41469-3

[8] Roy, G.; Brown, A.R.; Adamu-Lema, F.; Roy, S.; Asenov, A., "Simulation Study of Individual and Combined Sources of Intrinsic Parameter Fluctuations in Conventional Nano-MOSFETs," in Electron Devices, IEEE Transactions on , vol.53, no.12, pp.3063-3070, Dec. 2006.

[9] M. J. Orr. Introduction to radial basis function networks. Technical report, Centre for Cognitive Science, University of Edinburgh, 1996.

[10] Wikipedia. Gaussian elimination. https://en.wikipedia.org/wiki/Gaussian_elimination#Finding_the_inverse_of_a_matrix. [Accessed Dec. 2015]