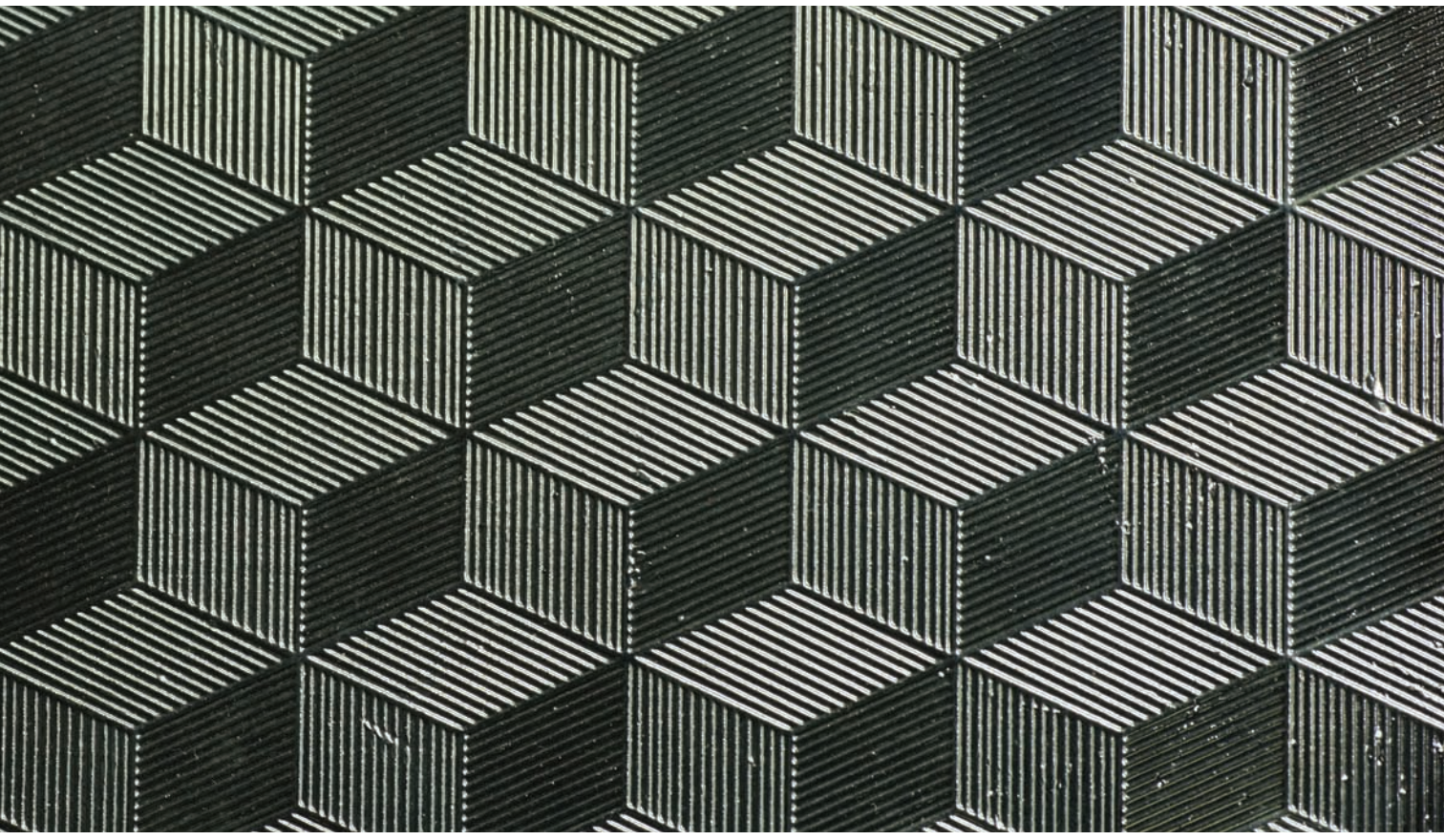
Editors: Barry I. Schneider, bis@nist.gov | Gabriel A. Wainer, gwainer@sce.carleton.ca



Using the Parallel DEVS Protocol for General Robust Simulation with Near Optimal Performance

Bernard P. Zeigler | University of Arizona and RTSync

The Discrete Event Systems Specification (DEVS) formalism has been widely disseminated in this magazine and elsewhere for its applicability to computational science and engineering. Bonaventura, Foguelman, and Castro[1] presented an iterative and incremental development methodology for simulation models in network engineering projects based on DEVS to assist network design, test, analysis, and optimization processes. Virtual Lab Environment (VLE) is a DEVS-based software environment to perform virtual experiments intended to increase the ability to cope with the complexity of living systems, enabling the choice of level of detail in models without being limited by the expressiveness of a single formalism.[2] Rhys Goldstein and Gabriel Wainer[3] showed how (particularly in the biological systems domain) DEVS provides a means of addressing complexity through hierarchical model design by illustrating the advantages of combining spatial decomposition in cellular automata with higher-level functional decomposition in Cell-DEVS.[4]

Nevertheless, the desire for fine-grained modeling and simulation (M&S) of biological and other systems continues to grow, demanding an ever-increasing ability to simulate complex models in reasonable time. DEVS-C++, a high-performance environment for modeling large-scale systems at high resolution, was shown to represent both continuous and discrete processes running in parallel with genetic

 1521-9615/17/$33.00 © 2017 IEEE Copublished by the IEEE CS and the AIP

Additional Reading on Parallel and Distributed Simulation

1. C.D. Christopher and K.S. Perumalla, "On Deciding between Conservative and Optimistic Approaches on Massively Parallel Platforms," *Proc. Winter Simulation Conf.*, 2010, pp. 678–687.
2. P.M. Dickens et al., "Analysis of Bounded Time Warp and Comparison with YAW NS," *ACM Trans. Modeling and Computational Simulation*, vol. 6, no. 4, 1996, pp. 297–320.
3. A. Gupta, I.F. Akyildiz, and R. Fujimoto, "Performance Analysis of Time Warp with Multiple Homogeneous Processors," *IEEE Trans. Software Eng.*, vol. 17, no. 10, 1991, pp. 1013–1027.
4. B.D. Lubachevsky, A. Weiss, and A. Shwartz, "An Analysis of Rollback-Based Simulation" *ACM Trans. Modeling and Computational Simulation*, vol. 1, no. 2, 1991, pp. 154–193.
5. D.M. Nicol, "Scalability, Locality, Partitioning and Synchronization," *Proc. Workshop Parallel and DistSimulation*, 1998, pp. 5–11.
6. D.M. Nicol, "Performance Bounds on Parallel Self-Initiating Discrete-Event Simulations," *ACM Trans. Modeling and Computational Simulation*, vol. 1, no. 1, 1991, pp. 24–50.
7. D.M. Nicol, "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations," *J. ACM*, vol. 40, no. 2, 1993, pp. 304–333.
8. F. Quaglia, F. Cortellessa, and B. Ciciani, "Trade-Off between Sequential and Time Warp-Based Parallel Simulation," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 8, 1999, pp. 781–794.
9. J. Zoltan et al., "A Performance Analyzer and Prediction Tool for Parallel Discrete Event Simulation," *Int'l J. Simulation Systems, Science & Technology*, vol. 4, no. 1, 2003, pp. 7–22.

Additional Reading on Amdahl's Law

1. K.W. Cameron and G. Rong, "Generalizing Amdahl's Law for Power and Energy," *Computer*, vol. 74, no. 3, 2012, pp. 75–77.
2. H. Che and M. Nguyen, "Amdahl's Law for Multithreaded Multicore Processors," *J. Parallel and Distributed Computing*, vol. 74, 2014, pp. 3056–3069.
3. M.D. Hill and M.R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, no. 7, 2008, pp. 33–38.
4. L. Kleinrock and J. Huang, "On Parallel Processing Systems: Amdahl's Law Generalized and Some Results on Optimal Design," *IEEE Trans. Software Eng.*, vol. 18, no. 5, 1992, pp. 434–447.
5. I. Onyuksel and S.H. Hosseini, "Amdahl's Law: A Generalization under Processor Failures," *IEEE Trans. Reliability*, vol. 44, no. 3, 1997, pp. 455–462.
6. A. Shah, "Scientist out to Break Amdahl's Law," 17 June 2013; www.pcworld.com/article/2042256/scientist-out-to-break-amdahls-law.html.
7. X.-H. Sun and Y. Chen, "Reevaluating Amdahl's Law in the Multicore Era," *J. Parallel and Distributed Computing*, vol. 70, no. 2, 2010, pp. 183–188.

algorithms.[5,6] While research in parallel and distributed simulation (PADS) has been active in the past several decades, the utility of many PADS techniques for high-performance simulation has been limited (see the "Additional Reading on Parallel and Distributed Simulation" sidebar). Such research typically starts from an event-oriented perspective of computation[7,8] from which it's often difficult, and in some cases impossible, to identify a priori the sequential and parallel parts of a model to which conventional parallelization techniques apply.

Recent research has shown that that a reconstruction of Amdahl's and Gustafson's laws for parallelizing sequential code can afford a better understanding of the underlying principles and their application to simulation of discrete event models, and DEVS models, in particular.[9,10] Gene Amdahl[11] asserted that a program can run no faster than the time it takes on a single processor divided by the number of processors (see the "Additional Reading on Amdahl's Law" sidebar). However, this speedup relation was stated without a proof based on first principles. John Gustafson[12] restated the law in terms more germane to distributed simulation such that the number of replications of the program that can be executed in the original time is limited by the number of processors. The recent formulation of the Amdahl/Gustafson speedup concepts for DEVS simulation affords new insights and an interpretation of the theory to parallel DEVS simulation.

In this article, I review some of the concepts relevant to the performance analysis of DEVS models and derive further implications for parallel DEVS simulations by summarizing the derivation of the Amdahl/Gustafson speedup concepts for DEVS simulations and discussing their application to the Parallel DEVS (PDEVS) simulation protocol. Noting that DEVS models represent the full class of discrete event dynamic systems, we can infer that the more incisive implications also apply more generally.

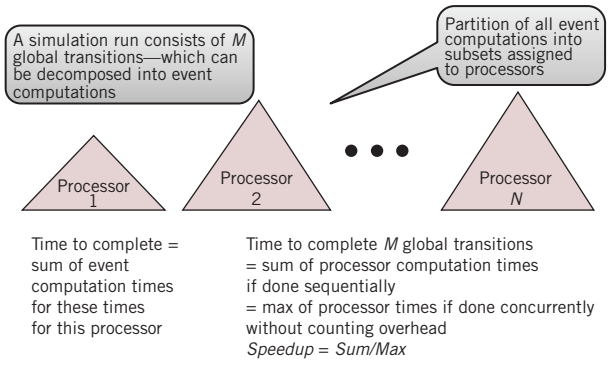


Figure 1. Partitioning of events among processors: the basic sum over max relation for computing speedup.

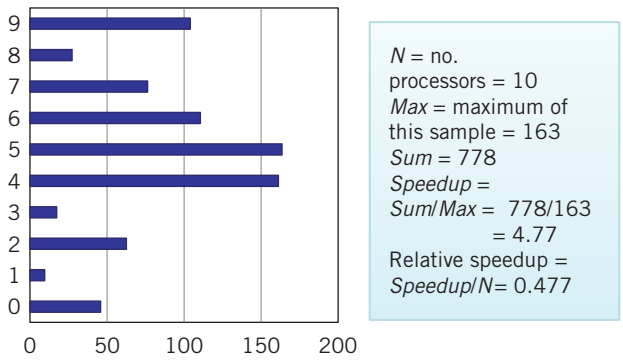


Figure 2. An example of an assignment of event computations to processors representing a distributed simulation. Note that the speedup (4.77) is less than the number of processors (10).

Reconstructing Gustafson's Law for Distributed Simulation

We can conceive of a logical (virtual) simulation execution as constituted by its events, whether internal (caused by itself) or external (caused by receipt of input). Executed on a single processor, the wall-clock time taken for this execution is defined as the sum of the individual event computation times over all the events. Now consider executing this simulation in a parallel and distributed manner, which amounts to partitioning the events among a number N of processors (nodes), as illustrated in Figure 1. We assume that the individual events take the same time to compute whether they're all on a single processor or within a smaller subset on a node processor. This is in keeping with the assumption that the original sequential simulation and the processors in the distributed simulation are all of the same platform type, having the same computation time for individual events. The fastest that any processor can execute all its events is the sum of their execution times.

In fact, the time it actually takes to execute this set could be larger than this minimum because it might have to wait for external events to arrive before it can proceed at various times. Different methods can have different characteristics in the additional wait and overheads incurred. However, since we're interested in the best that can be done, we can take the processor's time to complete its job (the assigned set of events) as the sum of its event computation times. Now, the time to complete the whole simulation is the maximum of these completion times since we must wait for all the processors to finish; the one, or ones, that takes the longest will determine the overall time. This leads to a definition of speedup for a distributed simulation on a network of processors as

$$Speedup = \frac{Sum}{Max}. \quad (1)$$

Here we first let CTi be the sum of the event computation times assigned to the ith processor, Sum is the sum of the CTi over all processors, and Max is the maximum of the CTi over all processors.

From the above discussion, Sum represents the time taken by the sequential simulation, while Max represents the time taken to execute the events in the largest partition block. Thus, the potential speedup of a partitioning of events is the sum of the processor times divided by the maximum of these times; see Figure 2 for an example.

Now we can state the equivalent of Gustafson's law for distributed simulation: the speedup of a simulation distributed on N processors can be no greater than N.

To see how this is true, notice that a sum of N non-negative numbers is less than or equal to N times the largest of these numbers. Using the above definitions, consider that

$$Sum = CT1 + CT2 + \ldots + CTN.$$

Dividing and multiplying by Max, we have

$$Sum = Max\left\{\frac{CT1}{Max} + \frac{CT2}{Max} + \ldots + \frac{CTN}{Max}\right\}.$$

And since each of the CTi terms is now less than or equal to 1, we have

$$Sum \leq Max \times N. \quad (2)$$

If we put Sum from Equation 2 in Equation 1 and simplify, we get

$$Speedup = \frac{Sum}{Max} \leq \frac{Max \times N}{Max} = N,$$

that is,

$$Speedup \leq N. \quad (3)$$

We see immediately that the best possible speedup is equal to the number of processors. This occurs when all processors have the same computation time. The sum in Equation 2 takes on its largest value when all of the terms take on the same value, the maximum. At that point, the speedup, namely, the *Sum*/*Max*, becomes equal to *N*.

Let the relative speedup be the actual speedup divided by the number of processors. Using the definition of speedup from Equation 1, and setting the average computation time *Avg* to *Sum*/*N*, we note that

$$Relative\ speedup = \frac{Speedup}{N} = \frac{Avg}{Max} \leq 1. \quad (4)$$

Indeed, since the average of a set of numbers is no greater than their maximum, relative speedup is a measure of how efficient an assignment of events to processors is. The distributed simulation law in this form effectively states that relative efficiency is at most one. My colleagues and I[9,10] have provided evidence for these observations:

- The relative speedup of a typical distributed simulation decreases with the number of processors; however, the absolute speedup could still increase—there's always likely to be an improvement in performance with an increasing number of processors (albeit with diminishing returns). This doesn't take communication delays into account.
- The greater the variation in processor computation times, the smaller the expected relative speedup and the faster the fall-off of relative speedup with increasing numbers of processors.
- When communication delays are accounted for, speedup could reach a peak and fall off from there with increasing processors.

Review of Modeling and Simulation Framework and DEVS

The modeling and simulation framework (MSF)[13] presents entities and relationships of a model and its simulation as background for the upcoming discussion (see Figure 3). The MSF separates models from simulators as entities that can be conceptually manipulated independently and then combined in a relation that defines correct simulation. The Experimental Frame defines a particular experimentation process, such as Latin hypercube sampling for yielding model outcome measurements in accordance with specific analysis objectives.

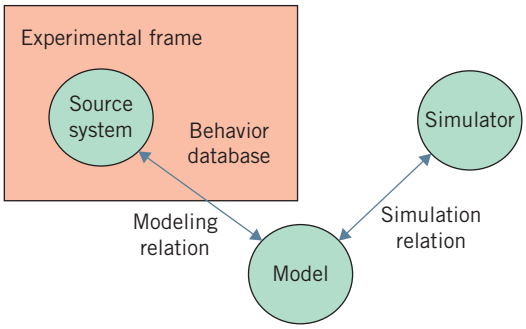


Figure 3. Modeling and simulation framework (MSF).

The DEVS formalism provides a sound and practical foundation for working with models and simulators. Briefly stated, a DEVS model is a system-theoretic concept specifying inputs, states, and outputs, similar to a state machine. Critically different, however, is that it includes a time-advance function that enables it to represent discrete event systems, as well as hybrids with continuous components in a straightforward platform-neutral manner.

A DEVS model is described by choices of sets of inputs, states, and outputs as well as functions that play crucial roles in defining its behavior: how it responds to inputs, changes states, and generates outputs over a continuous time base. At any time, such a model has a state s in set S. After an event, the simulator evaluates the time-advance function ta to schedule the internal event. Should this time elapse, the output function λ is invoked to obtain an output value y in Y, and the internal transition function δ_{int} yields a new state to replace the current state. If an input x in X is received before ta elapses, the simulator applies the external transition function δ_{ext} instead to obtain the new state. Briefly stated:

- DEVS formalizes what a model is, what it must contain, and what it doesn't contain (experimentation and simulation control parameters aren't contained in the model).
- DEVS is universal and unique for discrete event system models; any system that accepts events as

inputs over time and generates events as outputs over time is equivalent to a DEVS (its behavior and structure can be described by such a DEVS).
- DEVS-compliant simulators execute DEVS models correctly, repeatably, and efficiently. Closure under coupling guarantees correctness in hierarchical composition of components.
- DEVS models can be simulated on multiple different execution platforms, including those on desktops (for development) and those on high-performance platforms, such as multicore processors.

The MSF helps clarify many of the issues involved in M&S tasks. Mismatch between the simulation's time management policy and the model's time-advance approach creates significant errors in even the simplest M&S. Simulation with relatively coarse discrete time advance for a discrete event model exemplifies these kinds of errors. Distributed federations of discrete event and discrete time simulations with the high-level architecture (HLA),[14,15] are especially prone to conflicts between the intended, as-modeled event order and the implemented, as-simulated state-transition event order. Such conflict-based event causality errors in a tightly coupled simulation can introduce significant behavior deviations from the correct result. The MSF underlies the Parallel DEVS (PDEVS) simulation protocol, which provides provably correct simulation execution of DEVS models, thereby obviating the above-mentioned conflicts as well as throwing light on the source of such conflicts often found in simulations.

DEVS simulation is also distinguished by its support for both discrete event and continuous dynamic systems, both of which are simulated within the DEVS framework.[13] This capability to simulate the interaction of subsystems characterized by discrete event dynamics (such as communication networks and command-and-control systems) and continuous, physical dynamics (such as the trajectories of ballistic missiles and their interceptors) within the MSF makes DEVS particularly attractive to support distributed M&S.

Parallel DEVS Simulation Protocol

The PDEVS simulation protocol is a general distributed simulation protocol that prescribes specific mechanisms for

- declaring which component models take part in the simulation (component models);
- declaring how component models exchange data; and
- executing an iterative cycle that controls how time advances (time management), determines when component models exchange messages (data exchange management), and determines when component models do internal state updating (state update management).

The protocol guarantees correct simulation in the sense that if the component models are DEVS models, then the simulation result is also a well-defined DEVS coupled model. There are numerous implementations of DEVS simulators. Multiple conservative parallel discrete event simulation algorithms enable the parallel execution of DEVS models on these types of high-performance computing systems.[16,17] These algorithms are unique in the sense that they exactly reproduce the behavior of the DEVS reference simulator; this feature distinguishes DEVS from the numerous other conservative simulation engines that are derived from the logical process approach to PADS, which cannot reproduce the behavior of the DEVS reference simulator in all circumstances.

To see this, consider the PDEVS simulation protocol as outlined in Figure 4a's pseudo code. Each component DEVS model has a time of next event, t_N, that it sends to the coordinator, which calculates the minimum of these values, called the global t_N. Imminent components, whose t_N equals the minimum, compute their outputs and send them to receivers determined by the coupling specification. The active components, imminents and their receivers, then compute their state transition functions (internal, external, or confluent, depending on whether they're imminent or have inputs). Figure 4b illustrates the course of imminents over time as a simulation proceeds; four components' time advances are shown by successive arrows along the time line. A global transition occurs whenever at least one arrow tip is at the point representing a t_N. The plot at the bottom of Figure 4b shows the successive numbers of imminents with changes occurring at the global transitions.

In Figure 4a, lines 3 and 4 can each be executed in series or in parallel. For simplicity, for each component i, we'll lump the two into one event with a computation time, CTi, which is the sum of the times of the transition and output parts. Now for a sequential implementation, the time for a single global transition is the sum of the computation times. For a parallel implementation, the corresponding time is the maximum of component times: in the basic PDEVS protocol,

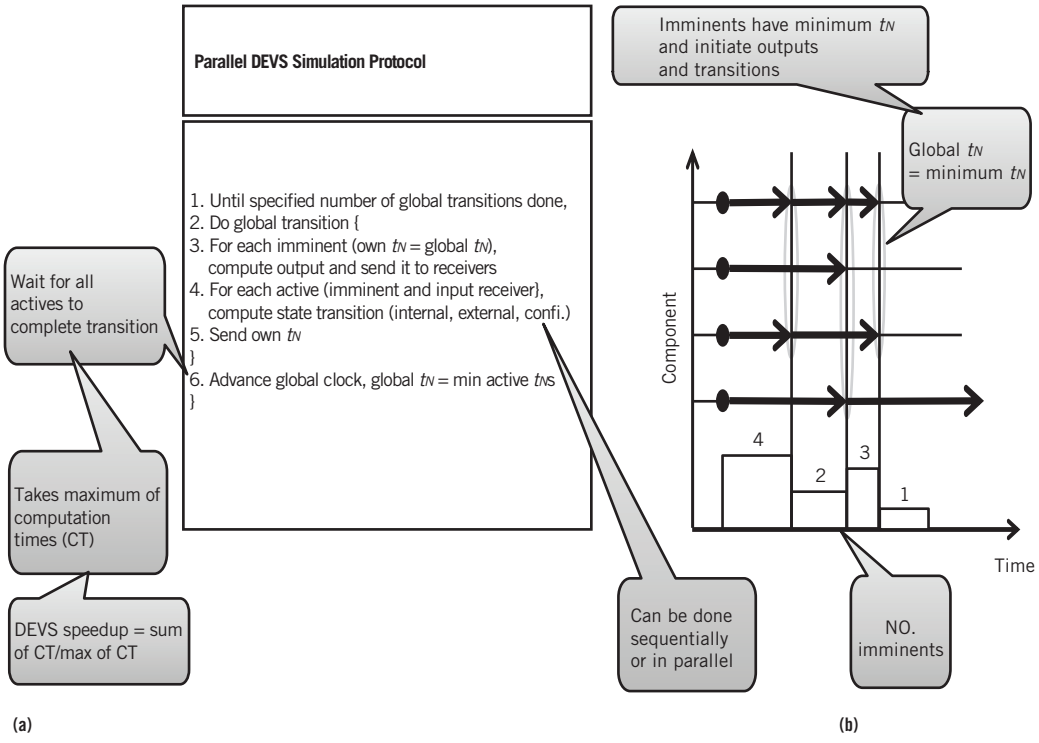


Figure 4. The PDEVS simulation protocol illustrating (a) the concept of imminent components and (b) the course of imminents over the time base as a simulation proceeds.

we must wait for each component to complete its transition before going to the next global transition.

Applying Speedup Concepts to DEVS Distributed Simulation

The concepts developed here enjoy more concrete instantiation when applied to distributed simulation of DEVS coupled models. The event sets in a partition described in Figure 1 can be identified with the internal and external events of DEVS component models in a coupled model. A simulation run is identified with a specified number, M, of global (coupled model) state transitions. In contrast to existing approaches,[18] our theory allows us to develop a first approximation or "back of the envelope" approach to predicting the best speedup that can be expected in the implementation process. The first approximation helps us understand the effects of the number of components and the distribution of simulation times of the components. We recap, and expand up, related work[9,10] for this approximation.

Equation 3 states that assigning each of the N component models to its own processor results in a speedup bounded by N. The *Sum* and *Max* of Equation 1 can be identified with the sum and maximum of the component runtimes, so that *Sum*/*Max* upper bounds the actual speedup. However, we can go deeper and characterize the parallelism exploited by the PDEVS simulation protocol by applying the approach iteratively to the M global transitions (Figure 4) and comparing this speedup with that derived for the M transitions taken as a whole. This lets us compare the potential speedup of the PDEVS protocol with the best possible speedup.

Equation 1 applies to each global transition, *GT*:

$$SumOfGT \leq MaxOfGT \times N, \tag{5}$$

where *CTiOfGT* is the sum of the transition and output computation times of the ith component, *SumOfGT* is the sum of the *CTiOfGT* over all components, and *MaxOfGT* is the max of the *CTiOfGT* over all components.

Summing the left and right sides of Equation 5 over all M global transitions, *GT*, and pulling N out, we have,

$$SumSumOfGT \leq SumMaxOfGT \times N, \tag{6}$$

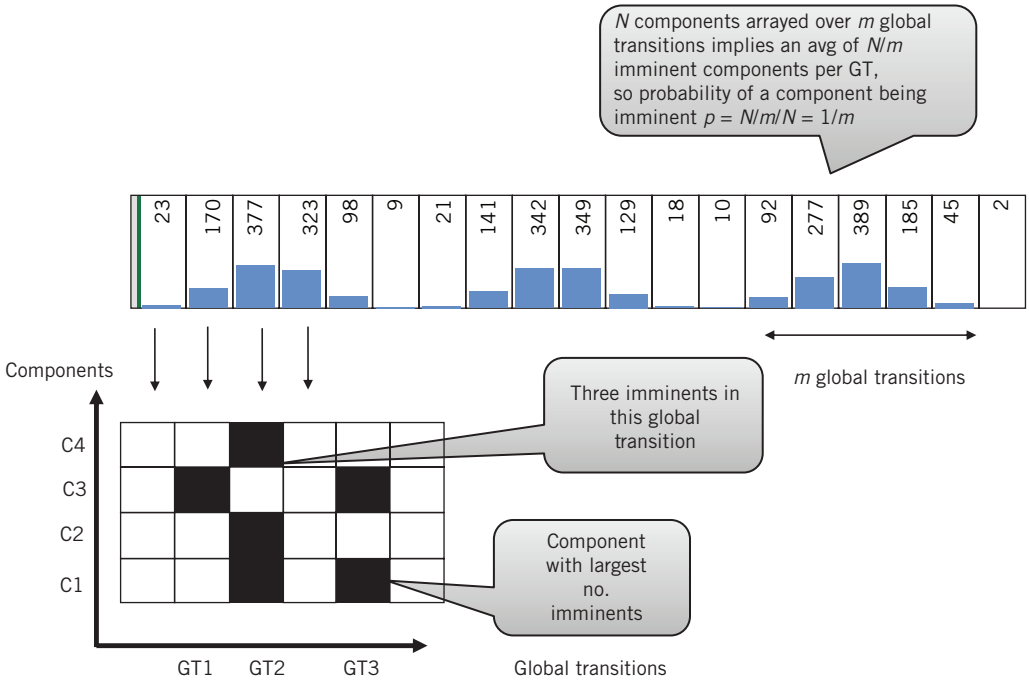


Figure 5. The checkerboard model on the bottom is a simple representation of a PDEVS simulation run involving *N* components (rows) through *M* global transitions (columns). Imminent components at global transitions are shown in black and passive in white. The parameter *p*, the probability of a component being imminent at a given transition, is derived from a trace of a run in which the numbers of imminents are recorded at each global transition. A repetitive pattern is shown in which all *N* components become imminent in the space of *m* transitions leading to a probability, $p = 1/m$.

where *SumSumOfGT* is the sum of all computation times for all *M* global transitions, and *SumMaxOfGT* is the sum of the maxima of each such transition with the time taken by the parallel version. Hence,

$$SpeedupOfPDEVS = \frac{SumSumOfGT}{SumMaxOfGT} \leq N. \qquad (7)$$

This is Gustafson's law reconstructed for the PDEVS protocol. We can compare it to a best speedup estimate for any method by assuming processing independence in the processors—that is, each processor is free to proceed at its own pace, and we wait for the slowest to finish. A direct application of Equation 3 yields

$$SpeedupOfInd = \frac{SumSumOfGT}{MaxSumOfGT} \leq N, \qquad (8)$$

where *MaxSumOfGT* is the maximum of the sum of each processor's computation times over the *M* transitions. Clearly the PDEVS protocol is no faster than the independent case. This verifies that our formulation reflects the fact that the PDEVS protocol assumes coupling exists among the components in contrast to the independence assumption where coupling is absent or ignored. Accordingly, we define the speed up of PDEVS relative to independence as

$$RelPDEVSSpeedup = \frac{MaxSumOfGT}{SumMaxOfGT} \leq 1, \qquad (9)$$

with inequality as shown.

To develop a simple stochastic model, we can reduce the situation to its essentials. Let all imminent computation times be ones and other times be zeros. Then, as in Figure 5 (bottom), *M* global transitions (columns) of *N* components (rows) can be visualized as a black and white image (assignment of zeros and ones to pixels.) Cell *i,j* represents the state of component *i* at global transition, *j*. Non-blank columns represent global transitions (where at least one component is imminent and shown as black), and each row represents the sequence of transition events experienced by a component.

Here,

$$SpeedupOfPDEVS = \frac{number\ of\ imminents}{number\ of\ GTs\ with\ imminents} = \frac{6}{3} = 2,$$

and

$$SpeedupOfInd = \frac{number\ of\ imminents}{Max\ number\ of\ imminents\ for\ components} = \frac{6}{2} = 3.$$

Such activity patterns can be analyzed for speedups.[19] For example, an all-black image represents a fully parallel coupled model where both speedups attain the maximum possible value, N.

To generate such patterns, we define a one-parameter stochastic model for which each cell is independently sampled with a probability p of being imminent. The probability can be estimated from traces of actual simulations or from a more refined model based on Monte Carlo executions of the PDEVS protocol in Figure 6. The version in the pseudo code there employs a granule of time to decide on component models whose t_Ns are within the global t_N as imminents. The larger the size of this granule, the greater the number of imminents is likely to be. Such a granule could exploit intrinsic temporal uncertainty in the model. Of course, this could also cause components that aren't truly simultaneous to be treated as such, requiring a tradeoff between performance and accuracy.[20]

The estimated probability of being imminent, P_{est}, is obtained by dividing the number of imminents counted in a run by the number of global transitions (to get the average number of imminents per transition) and then by the number of components N (to get the per component probability p) using the simple model. Empirical tests show that the estimated probability increases with granule size—specifically, Figure 5 shows that the time course of *#imminents* can become periodic as illustrated. Here, all N components become imminent within m global transitions in a repetitive manner. In this case, $P_{est} = 1/m$, as illustrated in the figure. Thus, the narrower the repetition period, m, the greater the estimated probability.

Figure 7 shows the results of computing relative speedups of PDEVS and independence protocols for probability values incremented in steps of .01. As expected, the independence curve always dominates the PDEVS curve except at $p = 1$. Accordingly, the speedup of PDEVS relative to independence rises

```
Set granule size, G, #components, N, and probability distribution, pdf
1. Until specified number of global transitions is reached,
2. Do global transition {
   Set #imminents = 0;
1. For each imminent (own t_N within G of global t_N) {
   #imminents = #imminents +1;
   sample t_N from pdf
   notify own t_N
       }
6. Advance global clock, global t_N = minimum of imminent t_Ns
```

Figure 6. Monte Carlo simulation model of the PDEVS protocol.

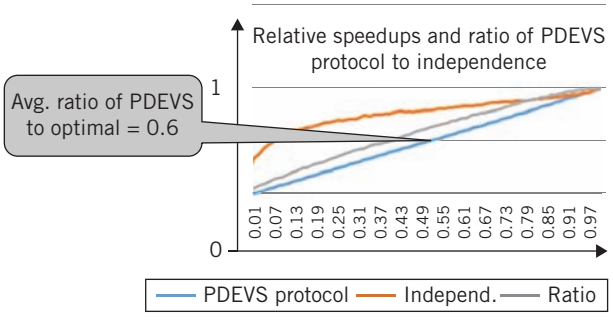


Figure 7. Relative speedups of PDEVS and independence protocols and their ratio (vertical axis) versus parameter p (horizontal axis).

from 0 to 1 with an average over all p values of 0.6. If we assume that actual simulations are distributed uniformly in p, then the expected speedup of the PDEVS protocol is 60 percent that of a fully independent protocol representing the best that any distributed simulation method can do.

We also added a second parameter, q, to model the effect of coupling on speedup. Here, q is the probability that an imminent component causes any given component to become immediately imminent (by sending it an input). Such new imminents have two effects: they increase the total number of imminents, and they increase the number of imminents any given component is likely to have, thus representing additional work due to interactions. Because they appear at the same time as existing imminents, these new ones don't affect the number of transitions with imminents. From Equations 8 and 9, the PDEVS speedup increases while that of the independent protocol decreases. Indeed, as displayed in Figure 8, we find that for the same conditions as before, the average speedup

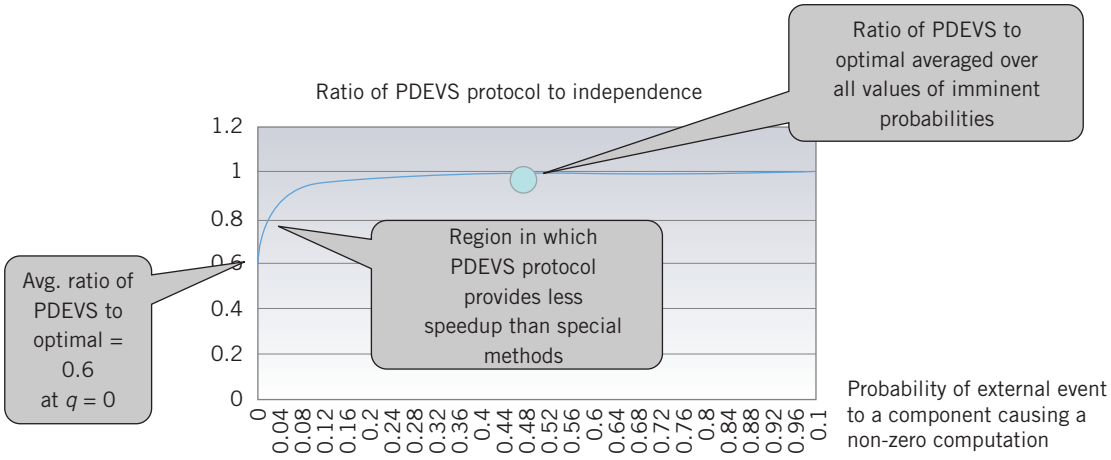


Figure 8. Ratio of PDEVS to optimal speedup averaged over p plotted against coupling probability, q.

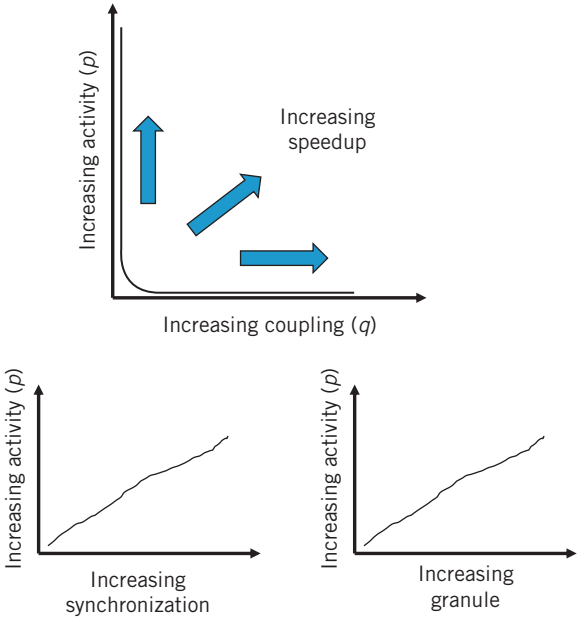


Figure 9. Summarizing the effects of activity and coupling on PDEVS speedup.

of PDEVS relative to independence now rapidly reaches 0.9 with $q \geq 0.06$. Therefore, a model with moderate coupling is likely to be just as well sped up by the PDEVS protocol as any other that might be used.

The PDEVS simulation protocol provides close to the best possible performance, except possibly where activity is very low or coupling among components is very small. Indeed, on average, the standard PDEVS protocol could require at most 60 percent more runtime than the best possible conservative or optimistic method. Moreover, the PDEVS protocol converges to the best possible speedup for models with moderate to high coupling. This is summarized in Figure 9, which illustrates that we can expect increasing speedup with increasing activity even without any coupling. On the other hand, coupling alone can't provide speedup without some activity to work on.

The effect of even very small amounts of coupling strongly amplifies activity and hence increases speedup. Activity is at its maximum when all components are synchronized to transition or output at the same time instance and continue to obey such synchronization, for example, in time-stepped cellular automata. Another way to increase activity is to use time granularization to increase the number of imminents at the next event.

Comparison with the best speedup that can be achieved indicates that the PDEVS protocol is a good, generally applicable approach to achieving speedup in parallel and distributed simulations. There might be particular circumstances in which more special conservative or optimistic methods might be warranted despite the extra work involved. However, typically, the extra performance gain is questionable given the additional cost, as well as the difficult-to-achieve conditions such as non-zero-lookahead that could be required. These findings suggest that simulator implementations focusing on repeatability and ease of use can replace relatively complicated and difficult-to-use parallel and distributed event simulation algorithms

that form the backbone of the HLA and other standards for constructive, distributed simulation. Research that tests the theory is required to verify the predications and validate the utility of the proposed model. ■

References

1. M. Bonaventura, D. Foguelman, and R. Castro, "Discrete Event Modeling and Simulation-Driven Engineering for the ATLAS Data Acquisition Network," *Computing in Science & Eng.,* vol. 18, no. 3, 2016, pp. 70–83.
2. G. Quesnel, R. Duboz, and E. Ramat, "The Virtual Laboratory Environment: An Operational Framework for Multi-modelling, Simulation and Analysis of Complex Dynamical Systems," *Simulation Modelling Practice and Theory,* vol. 17, 2009, pp. 641–653.
3. R. Goldstein and G. Wainer, "Designing Biological Simulation Models Using Formalism-Based Functional and Spatial Decompositions," *Computing in Science & Eng.,* vol. 17, no. 6, 2015, pp. 72–82.
4. R. Goldstein et al., "Vesicle-Synapsin Interactions Modeled with Cell-DEVS," *Proc. Winter Simulation Conf.,* 2008, pp. 813–821.
5. B.P. Zeigler et al., "DEVS Environment for High-Performance Modeling and Simulation," *IEEE Computational Science and Eng.,* vol. 4, no. 3, 1997, pp. 61–71.
6. D. Kim and B.P. Zeigler, "Orders of Magnitude Speedup with DEVS Representation and High Performance Simulation," *Proc. Enabling Technology for Simulation Science,* 1997; doi:10.1117/12.276715.
7. R.M. Fujimoto, "Parallel Discrete Event Simulation: Will the Field Survive?," *ORSA J. Computing,* vol. 5, no. 3, 1993, pp. 213–230.
8. E.H. Page, "Beyond Speedup: PADS, the HLA and Web-Based Simulation," *Proc. 13th Workshop Parallel and Distributed Simulation,* 1999, pp. 2–9.
9. B.P. Zeigler, J.J. Nutaro, and C. Seo, "What's the Best Possible Speedup Achievable in Distributed Simulation: Amdahl's Law Reconstructed," *Proc. DEVS TMS,* 2015; http://dl.acm.org/citation.cfm?id=2872991.
10. B.P. Zeigler and J.J. Nutaro, "Speedup Achievable in Distributed Simulation: Amdahl/Gufstafson's Law Reconstructed," submitted to *IEEE Trans. Parallel and Distributed Computing,* 2017.
11. G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proc. AFIPS Spring Joint Computer Conf.,* 1967; http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf.
12. J.L. Gustafson, " Reevaluating Amdahl's Law," *Comm. ACM,* vol. 31, no. 5, 1988, pp. 532–533.
13. B.P. Zeigler, H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation,* 2nd ed., Academic Press, 2000.
14. J. Nutaro, *Building Software for Simulation: Theory and Algorithms with Applications in C++,* Wiley, 2011.
15. J. Nutaro and H. Sarjoughian, "Speedup of a Sparse System Simulation," *Proc. 15th Workshop Parallel and Distributed Simulation,* 2001, pp. 193–199.
16. B. Cardoen et al., "A PDEVS Simulator Supporting Multiple Synchronization Protocols: Implementation and Performance Analysis," to appear in *Simulation,* 2017.
17. A. Adegoke, H. Togo, and M.K. Traoré, "A Unifying Framework for Specifying DEVS Parallel and Distributed Simulation Architectures," *Simulation,* vol. 89, no. 11, 2013, pp. 1293–1309.
18. S. Park, C.A. Hunt, and B.P. Zeigler, "Cost-Based Partitioning for Distributed and Parallel Simulation of Decomposable Multiscale Constructive Models," *Simulation,* vol. 82, no. 12, 2006; http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.302.6923&rep=rep1&type=pdf.
19. J.F. Santucci and L. Capocchi, "Implementation and Analysis of DEVS Activity-Tracking with DEVSimPy," *Proc. ACTIMS ITM Web Conf.,* 2013; doi:10.1051/itmconf/20130101001.
20. R.M. Fujimoto, "Exploiting Temporal Uncertainty in Parallel and Distributed Simulations," *Proc. Workshop Parallel and Distributed Simulations*, 1999, doi:10.1109/PADS.1999.766160.

Bernard P. Zeigler is an emeritus professor at the University of Arizona. His research interests include theory of modeling and simulation, parallel and distributed simulation, and model construction methodology. Zeigler has a PhD in computer and communication sciences from the University of Michigan. Contact him at zeigler@ece.arizona.edu.