



# Race conditions and data partitioning: risks posed by common errors to reproducible parallel simulations

James Nutaro and Ozgur Ozmen

## Abstract

When parallel algorithms for simulation were introduced in the 1970s, their development and use interested only experts in parallel computation. This circumstance changed as multi-core processors became commonplace, putting a parallel computer into the hands of every modeler. A natural outcome is growing interest in parallel simulation among persons not intimately familiar with parallel computing. At the same time, parallel simulation tools continue to be developed with the implicit assumption that the modeler is knowledgeable about parallel programming. The unintended consequence is a rapidly growing number of users of parallel simulation tools that are unlikely to recognize when the interaction of race conditions, partitioning strategies, and simultaneous action in their simulation models make results non-reproducible, thereby calling into question the validity of conclusions drawn from the simulation data. We illustrate the potential dangers of exposing parallel algorithms to users who are not experts in parallel computation with example models constructed using existing parallel simulation tools. By doing so, we hope to refocus tool developers on usability, even if this new focus incurs loss of some performance.

## Keywords

Parallel simulation, agent based, discrete event, reproducibility

## 1. Introduction

A principle of software design is *separation of concerns*, a term introduced by Dijkstra in 1974.<sup>1</sup> When discussing software for simulation, this concept manifests as a clear distinction between the model and its simulator. This separation recognizes that there are two distinct types of expertise needed to build a simulation model. The first is computational and mathematical: numerical methods, event schedulers, Lamport logical clocks, and so forth. The second is the domain of interest: epidemiology, psychology, electric circuits, and so forth. Simulation packages that separate model and simulator permit experts in the domain of interest to use the simulation tool without being experts in computation.

Simulation packages for the parallel execution of discrete event and agent-based models (ABMs) frequently violate this principle, to the detriment of repeatability and, potentially, the validity of a simulation model. The problem of creating reproducible simulations has many facets.<sup>2–5</sup> Here, we focus on the challenge of creating the software that embodies a simulation model<sup>6–9</sup> with a particular emphasis on the difficulties of creating correct parallel programs.

The parallel programming errors discussed here are not new, and technical solutions to these problems are well known. However, these solutions are not trivial. It is our position that the correct, parallel execution of a simulator is predicated on the design of its algorithms by an expert in parallel processing. Our modelers cannot be expected to have this expertise and so a separation of concerns is indispensable. Nonetheless, it has been our experience that the risks of violating this principle are often overlooked or, if acknowledged, are understated.

Historically, this separation of concerns has not been essential because parallel computers were relatively rare, and so expertise in parallel computation typically coincided with access to a parallel computer. This has changed over the past two decades. Now, parallel computers are

---

Computational Sciences and Engineering Division, Oak Ridge National Laboratory, USA

### Corresponding author:

James Nutaro, Computational Sciences and Engineering Division, Oak Ridge National Laboratory, 1 Bethel Valley Road, M.S. 6085, P.O. Box 2008, Oak Ridge, TN 37831, USA.

Email: nutarojj@ornl.gov

available to every modeler, but it does not follow that every modeler is an expert in parallel computing.

It has been demonstrated that the rudiments of parallel computing can be effectively taught to novice programmers;<sup>10</sup> and the brief survey therein, also.<sup>11</sup> Nonetheless, the complex parallel algorithms needed to resolve repeatability problems in many types of parallel simulations are not learned in a rudimentary course. For example, solutions to the problem of repeatably ordering simultaneous events in a parallel discrete event simulation involve the use of causality preserving logical clocks, a topic introduced only in advanced textbooks on parallel algorithms.<sup>12,13</sup>

The inherent difficulties of parallel algorithms ensure that most modelers are, essentially, sequential programmers except where powerful abstractions hide the complexities of a parallel computer. However, parallel simulation tools have historically emphasized performance and scale, and for this purpose frequently expose complex algorithmic machinery to the end user. The relatively new circumstance of complex parallel algorithms in the hands of sequential modelers poses new risks to model validity, which stem from the increasing likelihood that software for parallel simulation will be applied incorrectly.

The difficulties of programming parallel simulations are concretely demonstrated by reproducibility challenges that have come to light in prominent simulation programs from several domains of science and engineering, including computational fluid dynamics, traffic simulations, nuclear engineering,<sup>14–20</sup> and in various parallel simulation tools.<sup>14,15,21</sup>

While systematic studies of reproducibility are sometimes published for widely used simulation tools, such as those cited here, similar studies for models built and used by individuals or small teams are difficult to find. Nonetheless, if widely used simulations and simulation tools can exhibit unexpected non-reproducibility, it is reasonable to assume that these issues exist in other, less scrutinized parallel simulations.

At first glance, these reported issues are surprising. Simulation verification is expected to be part of the model development; hence, it is expected that any source of irreproducibility will be found and fixed. This view understates the difficulties of creating defect-free simulations<sup>22</sup> and, in the context of simulation libraries, the consequent importance of striving to avoid the possibility of defects arising from incorrect use.

This point is illustrated by Boehm and Basili<sup>23</sup> for software systems in general as follows:

About 80 percent of the defects come from 20 percent of the modules, and about half the modules are defect free. Studies from different environments over many years have shown, with amazing consistency, that between 60 and 90 percent of the defects arise from 20 percent of the modules ... *Obviously,*

*then, identifying the characteristics of error-prone modules in a particular environment can prove worthwhile.* A variety of context-dependent factors contribute to error-proneness. Some factors usually contribute to error-proneness regardless of context, however, including the level of data coupling and cohesion, size, *complexity*, and the amount of change to reused code. (Emphasis added).

It is our position that complexity, as seen from the modeler's perspective, is a characteristic of parallel simulation tools that make their use error prone. Hence, we suggest that the likelihood of errors leading to irreproducibility in parallel simulations can be reduced by reducing the complexity of the simulation tool as experienced by the modeler. That is, by separating the concerns of the modeler and the parallel programmer.

Solutions to the problem of reproducibility in parallel simulation programs are well known to experts in parallel simulation, but not necessarily to less experienced simulation programmers. A modeler that is experienced in parallel computing addresses the problems posed by parallel simulation in one of two ways. On one hand, the order in which a set of actions are applied is unimportant to the outcome; in this case, the computational overhead of imposing an order can be avoided without risk. Rao et al.<sup>24</sup> explore a version of this technique by discarding time synchronization to create a very fast queuing simulation, but their simulation requires an understanding of parallel computation to argue for its correct preservation of the model's statistical properties.

On the other hand, if the ordering of actions is significant, then specific mechanisms are put in place to enforce a proper order or to systematically explore alternative orderings.<sup>25–29</sup> This can involve complicated algorithms that require extensive knowledge and experience in parallel computation to apply correctly. Unfortunately, these solutions can entail a reduction of performance, and so they are often not included in the simulation package's default mode of operation.

Appeal to expertise is a satisfactory solution when experts are the primary users of parallel simulation tools, but it has become untenable for the increasingly diverse group of modelers interested in parallel simulation. To demonstrate some of the risks posed by an appeal to expertise when such expertise may be unavailable, we examine simple models that offer concrete examples of how validity can be undermined by errors unique to parallel computing. To avoid these risks, we propose that parallel simulation packages be designed to ensure repeatability by default, and that options to disable this property be exposed with suitable cautions to the user.

From the variety of errors that can appear in parallel simulation, we choose to illustrate two types that appear to be most prevalent and are unique to parallel programs: race conditions and incorrect partitioning. Discussions of these

errors, particularly in the context of new parallel programmers, are abundant.<sup>14–20</sup>

A race condition is an error that causes the ordering of actions to be a consequence of how threads and processes are scheduled rather than the logic of the simulation model. If left unresolved, race conditions in a simulation program can make it infeasible to both reproduce a single simulation execution and as well as recreate statistical distributions constructed from numerous simulation executions. This problem can appear even when parameter settings, experimental design, and executable software are identical to what was used before. Rather, any change in the computational environment, such as a change in the computer’s workload, an operating system update, or changed hardware is sufficient to alter the statistical properties of a parallel simulation model. Logical errors in the parts of a parallel program that perform partitioning and data exchange are another common problem. A study by Pederson<sup>30</sup> found that the tasks of data decomposition and function decomposition collectively constituted 25% of errors made by graduate students studying parallel computing in a one-semester course. In a parallel simulation program, these tasks involve distributing the model state—agents in an ABM and logical processes in a discrete event model—to threads and processes, and coordinating the exchange of data between the distributed agents or logical processes. A pernicious manifestation of these errors is that different results are obtained when the computational workload is distributed on a larger computer, even if any particular distribution of the workload yields a single result.

It is illuminating to compare the prevalent approach to parallel simulation with that of modeling and simulation techniques developed for real-time applications. Both pursuits are primarily concerned with creating a specific sequence of events in time. Parallel simulation, with its origins in high performance computing, seeks to do this as quickly and at as large a scale as possible. Modeling and simulation tools for the development of real-time systems are concerned with ensuring that simulation results reflect the target environment, striving for repeatability as an aide to debugging and performance analysis.<sup>31,32</sup>

The dissimilarity of emphasis in these two domains is noteworthy as both are vitally concerned with the ordering of events and their location in time. On one hand, in simulation tools and modeling methods used to develop real-time applications, we find an explicit concern with both the correctness of the application, as explored via simulation and, as a by product, an insistence that the simulation itself have predictable properties. At the same time, parallel simulation tools—and particularly discrete event and agent-based tools—accept the possibility of unpredictable behavior with the expectation that a knowledgeable user will recognize the circumstances in which surprises can

occur and have the skills needed to apply appropriate safeguards.

Our aim is to challenge this dissimilarity of expectation regarding the guarantees offered by a parallel simulation tool to the modeler. This challenge has two dimensions. The first has been to highlight the role that expertise in parallel computation has traditionally played in the successful use of parallel simulation tools, and to emphasize that this type of experience is becoming increasingly uncommon among the users of parallel simulation tools. This is a result of the combined effects of universal access to parallel computers and the growing importance of simulation to all fields of science and engineering.

The second aspect of our challenge is to show that even simple models may offer traps to the inexperienced parallel programmer. By falling into such a trap, the simulation model may become invalid by virtue of having outcomes dominated by computational artifacts. Worse, these invalid outcomes may manifest in skewed statistics rather than an obvious bug, and therefore be exceptionally difficult to detect. Our demonstrations illustrate the reproducibility problem at two levels: (1) L1: Deterministic, identical computation and (2) L2: Non-deterministic, identical computation.<sup>7</sup> These are the most commonly sought reproducibility levels, which demand repeatability for exact results (L1: deterministic) and statistical aggregations of results (L2: non-deterministic).

## 2. Race conditions

When a race condition is present in a parallel program, then the order of actions subject to the race condition is determined by the collective behaviors of the computational infrastructure: the operating system scheduler, communication network, memory caches, and so forth. Therefore, we should expect that the statistics of a simulation model containing a race condition will reflect the behavior of the computer system in some measure. For instance, suppose some aspect of our model concerns two outcomes  $A$  and  $B$ , and that our model assigns a probability  $p$  to  $A$  and  $1 - p$  to  $B$ . Further assume that a race condition in our parallel simulation model will choose  $A$  with probability  $q$  and  $B$  with probability  $1 - q$ .

Because a race condition does not always lead to an error, we will obtain some number of samples  $N$  due to the race condition and some number  $M$  due to the actual statistics of the model. An analysis of the data obtained will indicate that outcome  $A$  occurs with probability:

$$\hat{p} = \frac{qN + pM}{N + M}$$

As  $N + M$  becomes very large,  $N/(N + M)$  converges to a parameter  $\alpha$  and:

$$\hat{p} = \alpha q + (1 - \alpha)p$$

The parameters  $\alpha$  and  $q$  are artifacts of the computing system. They will depend on the number of processors and cores being used, design choices made by the operating system programmers, the computer's workload at the time the experiments are run, and other factors outside the control of the modeler. Consequently, we should not be surprised if our simulation results deviate from what would be obtained if the race condition was eliminated. Moreover, the deviation will change with the computational environment so that separate analysis using the same model may generate data that support conflicting conclusions.

A simple simulation model illustrates this effect. Consider the following ABM, which is implemented using C++ and OpenMP. This model has four agents, each with a single-state variable. At each time step, the state of agent  $i + 1$  is adopted by agent  $i$ . The space wraps around so that each agent has two neighbors. The intended order of execution for the agents is 1, 2, 3, and then, 4 and this behavior is realized by the simulator's single-threaded execution.

---

```
#include < iostream >
#include < unistd.h >
using namespace std;

int main() {
    // Four agents and their initial states
    int agent[4] = { 1, 0, 1, 0 };
    // Take three steps in time
    for (int t = 0; t < 3; t++) {
        // Print the current time
        cout << t;
        // This clause instructs the compiler
        // to execute the for loop iterations
        // in separate threads. The assignment
        // of loops to threads is determined by
        // the compiler, which will ensure the
        // iterates are divided as evenly as
        // possible among the available
        // threads.
        #pragma omp parallel for \
            ordered schedule(static,1)
        // Update the agents in order
        for (int i = 0; i < 4; i++) {
            // Input and output variables
            int x, y;
            // Do some work. This simulates a
            // calculation, database access,
            // or some other activity where
            // the effort is a function of
            // the agent identity.
            usleep(100*(3-i));
            // The agent output for the current
            // step
            y = agent[i];
            // The agent input for the current step
```

(continued)

```
x = agent((i + 1)%3]
// Print the output for the agents
// in order
#pragma omp ordered {
    cout << " " << y;
}
// Assign a new state to the agent
agent[i] = x;
}
// New line for the next time step
cout << endl;
}
}
```

---

If this simulation is executed using a single thread, the output is a translation of the initial state from left to right with the initial state reappearing at time 2.

```
0 1 0 1 0
1 0 1 0 1
2 1 0 1 0
```

If the simulation is executed using two threads, we obtain the same result every time the program is run, but this outcome is due to luck rather than design. It happens that agents 0 and 1 are assigned to the first thread, and agents 2 and 3 to the second thread. Because agents 2 and 3 need more time to perform their calculations, agents 0 and 1 perform their assignments to  $x$  and  $y$  first. A happenstance of scheduling means that the calculations happen in the correct order.

If the simulation is executed using three threads, a different result is obtained. Nonetheless, this new result is consistent for all executions that use three threads.

```
0 1 0 1 0
1 0 1 1 1
2 1 1 0 1
```

Once again, the outcome is a product of how agents are assigned to threads. The first thread is assigned agents 0 and 3, the second thread gets agent 1, and the third thread gets agent 2. As a result, the agents are updated in a different order with a different outcome. The outcome is persistent because large differences in the `usleep` times prevent a race between fetching  $x$  in one thread and assigning  $x$  in another.

Yet another result is obtained using four threads. In this case, each agent is assigned to its own thread. The interaction of the schedule and agent computations (here represented by `usleep`) ensures that the result is consistent across executions with four threads, and consistently different from executions using fewer threads.

```
0 1 0 1 0
1 0 1 1 0
2 1 1 0 1
```

The variety of results produced by the ABM are not due to an error or design flaw in OpenMP. Rather, they are the consequence of incorrectly using the parallel computing features that OpenMP offers. Indeed, this is the simplest possible case of statistical error introduced by a race condition. With two threads, we have  $\hat{p} = 1$  and the correct solution is always arrived at. With three or more threads,  $\hat{p} = 0$  and an incorrect solution is always produced. It so happens that our OpenMP implementation makes a consistent assignment of agents to threads, but this need not be the case. Should these assignments vary from run to run, then  $\hat{p}$  will reflect the frequency of assignments that yield a flawed result.

To demonstrate a model containing a race condition with  $\hat{p}$  far from 0 and 1, we use the Rensselaer’s Optimistic Simulation System (ROSS) parallel discrete event simulation tool.<sup>33</sup> As with the OpenMP example, the error in this model is due to our incorrect use of this (complex) parallel simulation tool and not some intrinsic fault of the tool itself. Nonetheless, to recognize and resolve the error requires some expertise in parallel computation.

This model has 100 logical processes. The state of each logical process comprises the number of events that it has executed, and a binary variable indicating which group of logical processes caused the execution of its most recent event. This variable takes a value of 0, if the most recent event was received from logical processes with identifier less than or equal to 50 and a value of 1 for a logical process with identifiers greater than 50.

Each logical process begins by scheduling an event for itself at time 1. For logical processes 1–99, this event generates a new event with time stamp 2 that is sent to logical process 0. Hence, logical process 0 will have 99 simultaneous events to execute at time 2. The result reported by this model is the state of logical process 0.

When this model is simulated using ROSS’s sequential and parallel conservative algorithms, the simulation algorithm produces the same result each time. However, a race condition inherent in the handling of simultaneous events prevents us from obtaining the same solution in each parallel run using ROSS’s optimistic algorithm. An analyst examining the data produced by the distinct simulation algorithms would conclude there are at least two models: one deterministic and the other stochastic.

Now suppose an analyst is supplied with five datasets produced in turn by executing a 1000 replications of the model using 2, 5, 10, 20, and 50 computational processes and the optimistic simulation procedure. In each case, our analyst computes the probability of obtaining a 1 from the simulation model. The result of this calculation is shown in Table 1. Because the means are very far apart in most cases, the exceptions being one and two computational processes, it is necessary to conclude there are five distinct models: one deterministic and four distinct, stochastic models.

**Table 1.** Probability of producing one.

Processes	Probability of 1	95% confidence interval
1,2	1	0
5	0.611	0.0308
10	0.708	0.0288
20	0.665	0.0299
50	0.577	0.0312

The role of  $\hat{p}$  in this example is clear. For the sequential execution and with two computation processes,  $\hat{p} = 1$  and the behavior is determined entirely by the logic of our model. For five computational processes,  $\hat{p} = 0.611$ ; for 10  $\hat{p} = 0.708$ , and so forth. In a larger model with more complicated behaviors, a novice parallel programmer may be unaware of the role played by the hardware and operating system in setting  $\hat{p}$  and, even more so, its degree of impact.

### 3. Incorrect partitioning

To demonstrate the types of errors that can be introduced by improper partitioning of a simulation model, we construct several ABMs using the Repast HPC simulation tool.<sup>34,35</sup> The errors introduced when partitioning the model are due to an improper use of boundary cells in Repast HPC and its impact on the order of agent execution. As before, to recognize and correct the error requires a non-trivial understanding of parallel computation.

#### 3.1. One-dimensional ABM

In this one-dimensional model, the agents are arranged in a line. The state of an agent is 0 or 1, and each agent can see the state of the agent to its right. At each time tick, each agent copies the adjacent agent’s state before updating its own.

In a sequential simulation, agent updates happen in a fixed order so that agent 1 acts first, then agent 2, and so on. Figure 1 presents the sequential execution of this model for four time ticks. Upon reaching time tick 4, agent 1 goes first, copies zero from agent 2, and all agent states remain at 0 for the rest of simulation time.

Our distributed memory implementation of this model realized with Repast HPC operates differently from the sequentially executing simulation, and yields different results depending on how the agents are assigned to computational processes. Specifically, the parallel algorithm creates what is called a grid projection with four cells, assigning one agent to each cell, and then distributing the cells among the available computational processes. In Repast HPC, the cell acts as a container, analogous to a physical space, in which an agent resides.

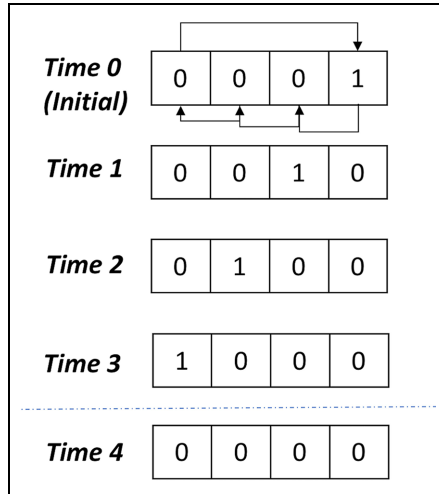


Figure 1. Expected deterministic outcome from the ID model.

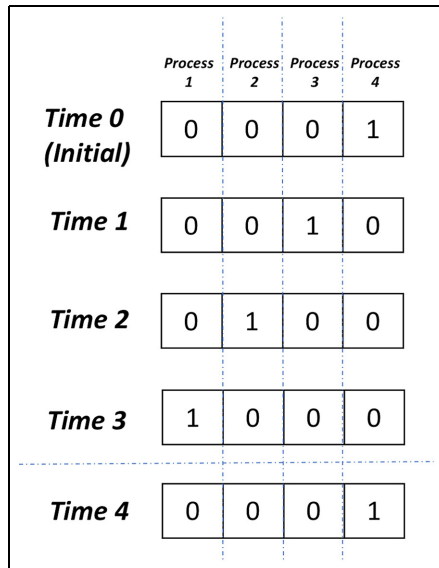


Figure 2. Parallel simulation with four computational processes in Repast HPC.

The state of a cell, and hence the state of the agents residing in that cell, are replicated across computational processes prior to executing a time tick. These replications are called buffer zones, and we add a one cell buffer zone between adjacent computational processes. The buffer zone causes a copy of the agent to the right to be available when executing the next time tick. Figure 2 shows the progress of this simulation over four time ticks when we use four computational processes for its execution.

The outcome is different from what is obtained with the sequentially executing model. At time tick 4, the agent in cell 4 has a copy of the agent in cell 1 (with state 1) and

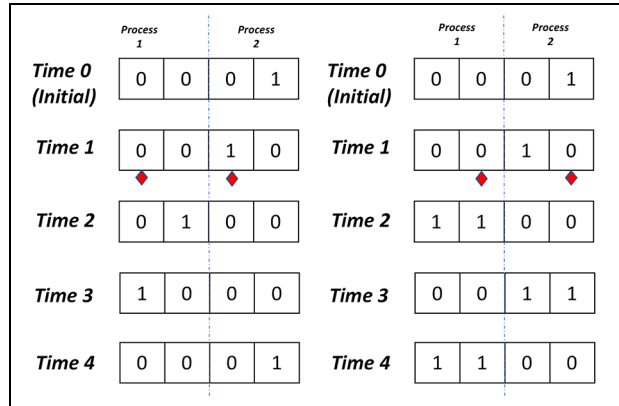


Figure 3. Parallelization with two processes in Repast HPC.

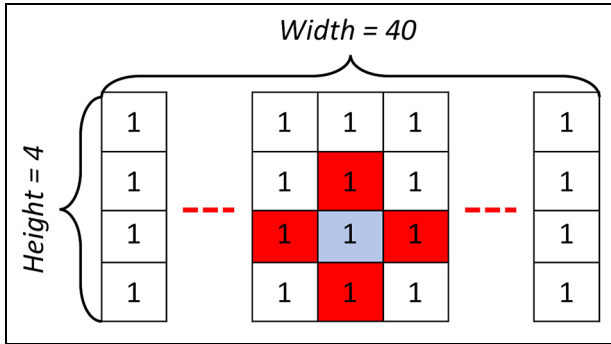
copies that state. The source of the discrepancy is the simultaneous updating of the agents using a copy of the state of the neighbor at the previous time tick. Those copies have no place in the sequentially executing model, are not a part of our original conception of the model, and so this is an artifact of the parallel algorithm.

To describe the impact of this type of parallel execution with Repast HPC better, we demonstrate two instances of the four-agent model distributed to two computational processes. This is done by creating two cells, each cell having two agents, and the adjacent cell in the neighboring computational process is copied at the beginning of each time tick. In one version of this simulation, the agents in each cell execute in turn from left to right and in the other from right to left. In both cases, the agents in each computational process are updated using copies of the adjacent cell state at the previous time tick.

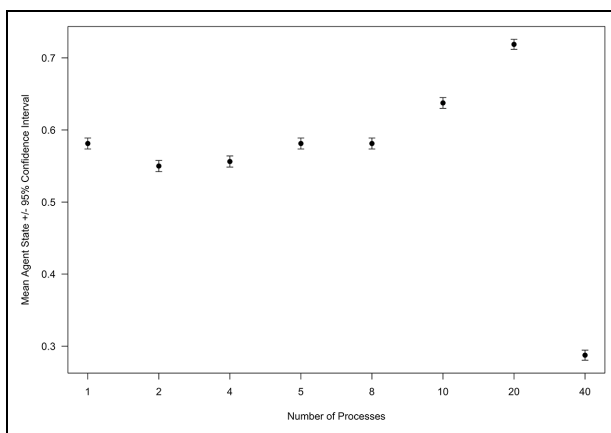
Both cases are shown in Figure 3 where diamonds indicate the agents that update first in each time tick. The result for the left to right ordering is the same as the four computational process execution in Figure 2. This is shown on the left-hand side of the figure. On the right, we observe the effect of a change in the update order.

### 3.2. Two-dimensional ABM

Now we construct a more typical model using Repast HPC by placing agents onto a two-dimensional grid. In this model, at each time tick, each agent observes its Von-Neumann neighborhood and changes its own state to the binary sum of its neighbors; i.e.,  $1 + 1 = 0$ ,  $0 + 1 = 1 + 0 = 1$ , and  $0 + 0 = 0$ . There is no random element in the summation process, and so an identical result should be obtained each time the simulation is run. In Figure 4, red cells show the Von-Neumann neighborhood. All agents start with a state of 1 and the model is executed for two time ticks.



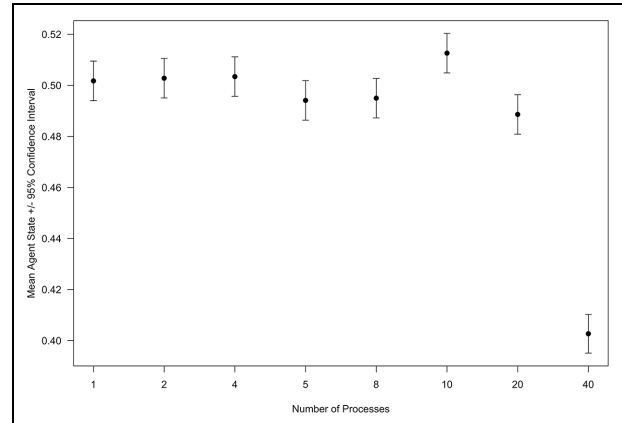
**Figure 4.** Initial state of a larger grid including 160 agents.



**Figure 5.** Error bars of agent states when the fixed model is run on different numbers of processes.

First, we consider a case where the initial conditions and order of agent updates is fixed for each computational process. When using multiple computational processes to execute this model, the grid is vertically divided into equal portions as in the Repast example with a single cell boundary at each computational process. Agents are updated in an order starting from the leftmost cell with the smallest index at each computational process. The model is run for two time ticks.

Figure 5 shows the mean and variance of agent states at the end of a simulation. Note that these calculations are at deterministic (L1) level and there is no need for replications. The variability in the results originate from the population of agents. The plot shows a clear change in mean and variance as the number of computational processes is increased, which indicates a qualitatively and quantitatively different result for each simulation. As before, this is an artifact of the boundary cells and the order of agent updates. These features have no place in the model as it was conceived for the sequentially executing simulator; they are an artifact of how we have used the parallel simulation features of Repast HPC.



**Figure 6.** Error bars of agent states for 100 replications when the shuffled model is run on different numbers of processes.

### 3.3. ABM with shuffling

So far, we demonstrated the artifacts of the partitioning on models with deterministic order of agent execution. However, when an ABM is implemented, the modeler is often indifferent to the order of agent updates, leaving the specific choice of ordering up to the underlying simulation procedure. In some tools, the scheduler scrambles the order of agent activation, and this may be the default choice for the simulator.<sup>36</sup> In a sequential simulation, when the model is run many times, the individual samples obtained using randomly selected update orders yield a genuine mean for the model. However, this is not necessarily true in a parallel simulation that does not explicitly address the question of agent order.

To demonstrate this, we let Repast HPC reshuffle the update order of our agents at each time tick for the model in Figure 4. This stochasticity means that the update order does not necessarily start from the leftmost agent and there is a different update order for each time tick. We ran 100 replications of the model on 1, 2, 4, 5, 8, 10, 20, and 40 computational processes using a fixed set of random number seeds for the scheduler. Each replication is run for two time ticks.

Figure 6 shows the mean and variance of the agent states for each case. These results represent stochastic (L2) level reproducibility. While 8 and 20 computational processes give statistically similar results, the case with 10 differs significantly. This clearly indicates that the statistics of the simulation output depend very strongly on the parallel computer.

To further illustrate the impact of the parallel computer on the simulation output, we compare the data for each case using a Wilcoxon test<sup>37</sup> in which the null hypothesis is that the data are generated by the sequentially executing model; that is, our intended model. The sequential and parallel models differ with very high confidence for the

**Table 2.** Wilcoxon test of agent states for different numbers of computational processes against results from a sequentially executing simulation.

Wilcoxon test	p-value	Parallel and sequential simulations are the same? (significance level 5%)
1 Process vs 2 Processes	0.8493	Accept
1 Process vs 4 Processes	0.7628	Accept
1 Process vs 5 Processes	0.1726	Accept
1 Process vs 8 Processes	0.2273	Accept
1 Process vs 10 Processes	0.0517	Accept
1 Process vs 20 Processes	0.0189	Reject
1 Process vs 40 Processes	< 2.2e-16	Reject

executions with 20 and 40 computational processes, and are only tentatively similar in other cases. The Wilcoxon test is summarized in Table 2.

#### 4. Conclusions and recommendations

The challenges to reproducibility that we have illustrated here are well known, and methods for ensuring reproducibility are likewise mature and widely employed by modelers with experience in parallel computing. For this reason, it has historically been true that strong guarantees of repeatability in parallel simulation tools were considered unimportant. The end user, presumed to be a skilled parallel programmer, would identify and resolve repeatability issues as needed without being hampered by the computational burden of a generalized solution that was unnecessary for the problem at hand.

This appeal to expertise has become impractical for the majority of users of parallel simulation tools. For a novice parallel programmer to use these solutions effectively, they must be intrinsic to parallel simulation tools and not an addition introduced as needed by the modeler. Toward this end, we offer a simple categorization of known solutions and examples within each category.

Solutions to the repeatability problem can be categorized into two bins. The solutions in the first bin violate Dijkstra’s principle of separating concerns by conflating the model and parallel simulation algorithm. These solutions should be avoided. Solutions in the second bin respect Dijkstra’s principle of separating concerns by separating the model and parallel simulation algorithm. Solutions in this bin should be preferred.

Solutions in the first bin are specific to a tool and are characterized by some exposure of the modeler to the solution mechanism. Although offering greater opportunities to optimize for performance, these solutions burden the modeler with parallel computing concerns and thereby violate the principle of separating concerns. Indeed, if

separation of concerns is indispensable to avoiding errors, then solutions in this bin are really not solutions at all. Two examples will illustrate this bin.

Our first example is Repast HPC and its ValueLayers feature. By using this feature, the simulation algorithm ensures that all agent states calculated at time tick  $t + 1$  use output values written to the value layer matrix at the end of time tick  $t$ . If this feature is used for our stochastic Repast model, then a consistent result is obtained regardless of how the model is partitioned among processes. However, this is not the default simulation approach. It must be deliberately chosen by the modeler, thereby exposing the modeler to the risk of non-reproducible results.

Our second example is the capability in ROSS for the modeler to define a type that represents time. It is possible to define a multi-dimensional form of time in such a way that a unique ordering of events emerges. Examples of this appear in WarpIV<sup>38</sup> and several other parallel discrete event simulation tools.<sup>13</sup> A generalized form of this concept is super-dense time as it appears in simulations of hybrid dynamic and discrete event systems.<sup>39</sup> Correct use of these solution techniques within ROSS would require the modeler to have a working understanding of these forms of simulation time, thereby exposing the modeler to the risk of non-reproducible results.

In the second bin of solutions are simulation tools offering a model specification framework that is mathematically well-defined and may be correctly realized in software by a variety of means.<sup>40-43</sup> This approach is analogous to the distinction between models specified with differential equations and the numerical methods used to solve those equations. A concise, well-defined boundary between the model and its simulator satisfies the principle of separating concerns. Two examples from this bin are the Model of Computation approach pioneered by Edward Lee<sup>44,45</sup> and the Discrete Event System Specification (DEVS) introduced by Bernard Zeigler et al.<sup>40</sup>

Lee describes a model of computation as “the set of *laws of physics* that govern the interaction of components in the model. If the model is describing a mechanical system, then the model of computation may literally be the laws of physics. More commonly, however, it is a set of rules that are more abstract, and provide a framework within which a designer builds models. A set of rules that govern the interaction of components is called the semantics of the model of computation.”

The modeler need only be concerned with these semantics, and simulation tools realizing the semantics will produce the same results even though their software implementations may be different. For complicated models, several distinct models of computation may be composed. Like individual model components, a composite may be understood and used by the modeler without knowledge of how the composition is realized in software by the underlying simulation tool.



The DEVS approach also offers semantics for model construction that are invariant with respect to how the simulation tool is implemented. In broad outlines, DEVS differs from the Model of Computation approach by seeking generality rather than specificity; a single semantics universally applied rather than a composition of specialized semantics.

There are many simulation tools that implement the DEVS semantics,<sup>46,47</sup> and these disparate tools demonstrate the reproducibility can be achieved within a tool and across tools when used to implement the same model. For example, benchmark models are reproduced precisely across simulator implementations to compare performance<sup>48</sup> and it is feasible to reuse component models across simulator implementations.<sup>49</sup> Moreover, these reproducible results extend naturally to DEVS tools implemented using parallel discrete event simulation algorithms; e.g., see the simulation algorithms presented in the study by Nutaro.<sup>50,51</sup>

Despite the availability of parallel simulation packages that incorporate solutions in the second bin, these presently constitute a small part of the parallel simulation landscape. In our view, this situation is a legacy of the historical emphasis on speed and scale in parallel simulation rather than usability. Indeed, the expertise needed to apply parallel discrete event simulation was recognized very early in that field's history as a barrier to its adoption by practitioners.<sup>52</sup>

Now, powerful parallel computers are readily available and it is natural for every modeler to want to use a parallel computer to speed up their increasingly sophisticated models. At the same time, the difficulty of correctly programming a parallel computer creates new risks that discoveries made via simulation studies will not be reproducible, and so not meaningfully contribute to the advancement of science and technology. It is impractical to require modelers that use a parallel computer to also master the intricacies of parallel simulation. Instead, tool builders who are experts in parallel simulation must strive to create environments that limit or eliminate these risks for the modeler, whose expertise lies in the domain of study rather than the technology of parallel simulation.

## 5. Code repositories

Source codes for the ROSS and Repast HPC models are available in the repositories below:

- <https://ozi@code.ornl.gov/ozi/parallelSIMDemos.git>

Please contact Ozgur Ozmen ([ozmeno@ornl.gov](mailto:ozmeno@ornl.gov)) if you have difficulties connecting to the repository or need to be granted access privileges to it. Users from outside of Oak Ridge National Laboratory (ORNL) can connect to the

repository by creating an XCAMS account (<https://xcams.ornl.gov>) and logging in with that account on <https://code.ornl.gov>.

- <https://github.com/cengover/parallelABMs.git>


## Acknowledgements


This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of the manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

## ORCID iDs

James Nutaro  <https://orcid.org/0000-0001-7360-2836>

Ozgur Ozmen  <https://orcid.org/0000-0002-5806-0116>

## References

1. Dijkstra EW. *Selected writings on computing: a personal perspective*. Berlin: Springer, 1982.
2. Yilmaz L. Reproducibility in M&S research: issues, strategies and implications for model development environments. *J Exp Theoret Artif Intell* 2012; 24: 457–474.
3. Yilmaz L, Taylor S, Fujimoto R, et al. Panel: the future of research in modeling & simulation. In: *Proceedings of the 2014 winter simulation conference*, Savannah, GA, 7–10 December 2014, pp. 2797–2811. Piscataway, NJ: IEEE.
4. Donkin E, Dennis P, Ustalakov A, et al. Replicating complex agent based models, a formidable task. *Environ Model Software* 2017; 92: 142–151.
5. Uhrmacher AM, Brailsford S, Liu J, et al. Reproducible research in discrete event simulation: a must or rather a maybe. In: *Proceedings of the 2016 winter simulation conference (WSC '16)*, Washington, DC, 11–14 December 2016, pp. 1301–1315. New York: IEEE.
6. Fitzpatrick BG. Issues in reproducible simulation research. *Bull Math Biol* 2019; 81: 1–6.
7. Dalle O. On reproducibility and traceability of simulations. In: *Proceedings of the 2012 winter simulation conference (WSC)*, 2012, pp. 1–12, <https://hal.inria.fr/hal-00782834/document>

8. Galán JM, Izquierdo LR, Izquierdo SS, et al. Errors and artefacts in agent-based modelling. *J Artif Soc Soc Simul* 2009; 12: 1.
9. Wilensky U and Rand W. Making models match: replicating an agent-based model. *J Artif Soc Soc Simul* 2007; 10: 2.
10. Conte DJ, de Souza PSL, Martins G, et al. Teaching parallel programming for beginners in computer science. In: *2020 IEEE frontiers in education conference (FIE)*, 2020, pp. <https://aic-atlas.s3.eu-north-1.amazonaws.com/projects/e7299991-cb2b-4764-a849-4909e01fb07d/documents/iQ2aDCshOAsjNsFHMLYOizMKALLtTBTQCVB17SIh.pdf> 1–9.
11. Blandin N, Colglazier C, O'Hare J, et al. Parallel python for agent-based modeling at a global scale. In: *Proceedings of the 2017 international conference of the computational social science society of the Americas*, Santa Fe, NM, October 19–22, 2017, pp. 1–7. New York: Association for Computing Machinery.
12. Garg VK. *Principles of distributed systems*, vol. 3144. Berlin: Springer, 2012.
13. Ronngren R and Liljenstam M. On event ordering in parallel discrete event simulation. In: *Proceedings thirteenth workshop on parallel and distributed simulation (PADS 99) (Cat. No. PR00155)*, Atlanta, GA, 1–4 May 1999, pp. 38–45. New York: IEEE.
14. Keum S, Grover R Jr, Gao J, et al. Effect of parallel computing environment on the solution consistency of CFD simulations -focused on IC engines. *Engineering* 2017; 9: 824–847.
15. Nheili R, Langlois P and Denis C. First improvements toward a reproducible Telemac-2D. In: *Proceedings of the XXIIIrd TELEMAR-MASCARET user conference*, Paris, 11–13 October 2016, pp. 227–235. New York: IEEE.
16. Hill DRC. Repeatability, reproducibility, computer science and high performance computing: stochastic simulations can be reproducible too. In: *2019 international conference on high performance computing simulation (HPCS)*, Dublin, 15–19 July 2019; pp. 322–323. New York: IEEE.
17. Diethelm K. The limits of reproducibility in numerical simulation. *Comput Sci Eng* 2011; 14: 64–72.
18. Hill DR. Parallel random numbers, simulation, and reproducible research. *Comput Sci Eng* 2015; 17: 66–71.
19. Avery C. *Scalable, repeatable, and contention-free parallelization of traffic simulation*. Cambridge, MA: Massachusetts Institute of Technology, 2018.
20. Reuillon R, Hill DR, El Bitar Z, et al. Rigorous distribution of stochastic simulations using the DistMe toolkit. *IEEE Trans Nuclear Sci* 2008; 55: 595–603.
21. Maigne L, Hill D, Calvat P, et al. Parallelization of Monte Carlo simulations and submission to a grid environment. *Parallel Process Lett* 2004; 14: 177–196.
22. Robinson S. Simulation model verification and validation: increasing the users' confidence. In: *Proceedings of the 29th conference on winter simulation (WSC '97)*, 1997, p. 5359. New York: IEEE, <https://www.informs-sim.org/wsc97papers/0053.PDF>
23. Boehm B and Basili VR. Software defect reduction top 10 list. *Computer* 2001; 34: 135137.
24. Rao DM, Thondugulam NV, Radhakrishnan R, et al. Unsynchronized parallel discrete event simulation. In: *Proceedings of the 30th conference on winter simulation*, 1998, pp. 1563–1570. Los Alamitos, CA: IEEE. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.8404&rep=rep1&type=pdf>
25. Peschlow P and Martini P. Efficient analysis of simultaneous events in distributed simulation. In: *11th IEEE international symposium on distributed simulation and real-time applications (DS-RT 2007)*, Chania, 22–26 October 2007, pp. 244–251. Piscataway, NJ: IEEE.
26. Barz C, Gopffarth R, Martini P, et al. A new framework for the analysis of simultaneous events. In: *Proceedings of summer computer simulation conference*, 2003, pp. 306–313. San Diego, CA: Society for Computer Simulation, <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.496.1044&rep=rep1&type=pdf>
27. Nutaro J and Sarjoughian H. Design of distributed simulation environments: a unified system-theoretic and logical processes approach. *Simulation* 2004; 80: 577–589.
28. Fujimoto RM. *Parallel and distributed simulation systems*. Hoboken, NJ: John Wiley & Sons, 2000.
29. Chow ACH and Zeigler BP. Parallel DEVS: a parallel hierarchical modular modeling formalism. In: *Proceedings of winter simulation conference*, 1994, pp. 716–722, [http://www.bgc-jena.mpg.de/~twutz/devsbridge/pub/chow96\\_parallelDEVs.pdf](http://www.bgc-jena.mpg.de/~twutz/devsbridge/pub/chow96_parallelDEVs.pdf)
30. Pedersen JB. Classification of programming errors in parallel message passing systems. In: *Communicating process architectures*, 2006, pp. 363–376, <http://www.egr.unlv.edu/~matt/publications/pdf/CPA-2006.pdf>
31. Niyonkuru D and Wainer G. A DEVS-based engine for building digital quadruplets. *Simulation* 2021; 97: 485–506.
32. Samuel KG, Bouare NDM, Maïga O, et al. A DEVS-based pivotal modeling formalism and its verification and validation framework. *Simulation* 2020; 96: 969–992.
33. Carothers CD, Bauer D and Pearce S. ROSS: a high-performance, low-memory, modular Time Warp system. *J Parallel Distribut Comput* 2002; 62: 1648–1669.
34. Collier N and North M. Repast HPC: a platform for large-scale agent-based modeling. *Large-Scale Comput* 2012; 10: 81–109.
35. Collier N and North M. Parallel agent-based simulation with Repast for high performance computing. *Simulation* 2013; 89: 1215–1235.
36. North MJ and Macal CM. Product design patterns for agent-based modeling. In: *Proceedings of the 2011 winter simulation conference (WSC)*, Phoenix, AZ, 11–14 December 2011, pp. 3082–3093. New York: IEEE.
37. Wilcoxon F. Individual comparisons by ranking methods. *Biometrics Bull* 1945; 1: 80–83.
38. Steinman JS. The WarpIV simulation kernel. In: *Workshop on principles of advanced and distributed simulation (PADS'05)*, 2005, pp. 161–170, <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.516.9749&rep=rep1&type=pdf>
39. Nutaro J. Toward a theory of superdense time in simulation models. *ACM Trans Model Comput Simul* 2020; 30(3): 3379489.
40. Zeigler B, Muzy A and Kofman E. *Theory of modeling and simulation*. 3rd ed. London: Academic Press, 2019.

41. Grimm V, Railsback SF, Vincenot CE, et al. The ODD protocol for describing agent-based and other simulation models: a second update to improve clarity, replication, and structural realism. *Journal of Artificial Societies and Social Simulation* 2020; 23: 7.
42. Brooks C, Lee EA and Tripakis S. Exploring models of computation with Ptolemy II. In: *2010 IEEE/ACM/IFIP international conference on hardware/software co-design and system synthesis (CODES + ISSS)*, 2010, pp. 331–332, <https://ptolemy.berkeley.edu/projects/chess/pubs/712/PTutorialHandout.pdf>
43. Lee EA and Zheng H. Operational semantics of hybrid systems. In: Morari M and Thiele L (eds) *Proceedings of the 8th international workshop on hybrid systems: computation and control, HSCC 2005*. Berlin; Heidelberg: Springer, 2005, pp. 25–53.
44. Lee EA. *Overview of the Ptolemy project*. Berkeley, CA: EECS Department, University of California, Berkeley, 1998.
45. Ptolemaeus C (ed.). *System design, modeling, and simulation using Ptolemy II*, 2014, <http://ptolemy.org/books/Systems>
46. Franceschini R, Bisgambiglia PA, Touraille L, et al. A survey of modelling and simulation software frameworks using Discrete Event System Specification. In: Neykova R and Ng N (eds) *2014 Imperial College Computing Student Workshop. Vol. 43 of Open Access Series in Informatics (OASICS)*. Wadern: Schloss Dagstuhl–Leibniz-Zentrum Fuer Informatik, 2014, pp. 40–49.
47. Van Tendeloo Y and Vangheluwe H. An evaluation of DEVS simulation tools. *Simulation* 2017; 93: 103121.
48. Risco-Martn JL, Mittal S, Fabero J, et al. Reconsidering the performance of DEVS modeling and simulation environments using the DEVStone benchmark. *Simulation* 2017; 93: 459–476.
49. Wutzler T and Sarjoughian HS. Interoperability among parallel DEVS simulators and models implemented in multiple programming languages. *Simulation* 2007; 83: 473–490.
50. Nutaro JJ. *Building software for simulation: theory and algorithms with applications in C++*. Hoboken, NJ: John Wiley & Son, 2011.
51. Nutaro JJ. On constructing optimistic simulation algorithms for the discrete event system specification. *ACM Trans Model Comput Simul* 2009; 19: 1.
52. Fujimoto RM. Feature article parallel discrete event simulation: will the field survive? *INFORMS J Comput* 1993; 5: 213–230.

### Author biographies

**James Nutaro** is a Distinguished Research and Development Scientist in the Computational Sciences and Engineering Division of Oak Ridge National Laboratories, where he leads the Computational Systems Engineering & Cybernetics group. Dr Nutaro is engaged in research concerning discrete event and hybrid dynamic systems, modeling and simulation methods, and their applications to problems in systems engineering. He holds a PhD in Computer Engineering from the University of Arizona. His email address is [nutarobjj@ornl.gov](mailto:nutarobjj@ornl.gov).

**Ozgur Ozmen** is a Research and Development Staff in the Computational Sciences and Engineering division of the Oak Ridge National laboratory. He holds an Industrial Engineering degree from Yildiz Technical University in Istanbul, Master of Engineering Management degree from Galatasaray University, MISE, and PhD Degrees (both in Industrial and Systems Engineering) from Auburn University. He is a model agnostic interdisciplinary engineer who applied Modeling and Simulation, optimization, and Machine learning to wide variety of domains such as Energy systems, Operational Research, and Healthcare.