Computational Materials Science 219 (2023) 111990

Contents lists available at ScienceDirect

Computational Materials Science

journal homepage: www.elsevier.com/locate/commatsci

Full length article

Discrete event cellular automata: A new approach to cellular automata for computational material science☆

James Nutaro *, Benjamin Stump, Pratishtha Shukla

Oak Ridge National Laboratory, One Bethel Valley Road, Oak Ridge, 37830, TN, USA

ARTICLE INFO

Keywords:
Cellular automata
Discrete event simulation
Grain growth

ABSTRACT

We explore the computational advantages of discrete event simulation for cellular automata models of grain growth. These benefits include a reduction in execution time by up to an order of magnitude and the elimination of numerical errors that stem from overshooting grain capture events and approximating a Poisson process with a Bernoulli process. The fundamental mechanisms speeding up the discrete event simulation are uncovered, and with these we create a speedup model that explains our experimental outcomes.

1. Introduction

Our aim is to introduce discrete event simulation as an efficient and numerically accurate method of computation for cellular automata models of grain growth. For this purpose, we develop two simple, but pertinent, simulators for two well known models. Our first simulator realizes the probabilistic cellular automaton introduced by Raabe [1,2] in a discrete event form. This cellular automaton models rate of growth with a transition probability, which constitutes a Bernoulli process if the computation proceeds in fixed steps. As the step size tends towards zero, this Bernoulli process tends towards a Poisson process. With this example, we show how a discrete event simulation realizes this model in its limit (i.e., as a Poisson process), thereby eliminating numerical errors in the Bernoulli approximation. At the same time, we demonstrate a speedup over the time stepped model which increases as the step size for the time stepped model shrinks.

Our second simulation is a discrete event realization of the decentered square model of grain growth [3,4]. This model is typically realized with a discrete time simulation, for which a time step must be selected. A large time step improves the execution time at the cost of increased error, which manifests in the form of simultaneous capture events that would not occur in the physical process. Smaller step sizes reduce the chance for these events to occur, but incur a greater computational cost. Our discrete event simulation eliminates these simultaneous capture events, and thereby eliminates the errors that these introduce. We compare the execution times of the discrete event and discrete time simulators when the latter uses a step size that is small enough to eliminate simultaneous capture events.

Over the last two decades there have been several algorithmic advancements over earlier forms of cellular automata proposed specifically for materials problems. Some of these advancements have taken steps towards a discrete event form of simulation. For example, front tracking methods reduce time to solution by focusing computational effort on the dynamic portion of the cell space, avoiding unnecessary computations in areas away from the grain boundary [5]. A similar proposal that cells be separated into active and inactive sets seeks to accomplish the same aim in a more general space [6]. Distinct rates of evolution across the cell space are identified by the algorithm proposed in [7], which is a natural first step towards the time advance in a discrete event model.

Discrete event simulation can be seen as integrating these several ideas into a single framework for computation. Indeed, the motivations cited in each case strive towards a specialized form of activity tracking, which describes the natural tendency of a discrete event simulator to focus computational effort on components in proportion to their rate of evolution [8,9]. Our example models show how the discrete event simulation algorithms that realize activity tracking [10–15] can be

* Corresponding author.
E-mail address: nutarojj@ornl.gov (J. Nutaro).

[1] The software used to perform our simulation experiments is available as the grain growth example that comes with the adevs simulation package at https://sourceforge.net/projects/adevs/.

https://doi.org/10.1016/j.commatsci.2022.111990
Received 17 June 2022; Received in revised form 18 November 2022; Accepted 13 December 2022
Available online 5 January 2023

applied to cellular automata in the context of computational material science.

We begin by outlining the primary elements of a discrete event model and its simulation procedure. Our approach is based on the Discrete Event System Specification (DEVS) [16], for which a variety of open source simulation tools are available.[1] After this introduction, we describe the key steps for transforming a discrete time model into a discrete event model. Afterwards, the two example models are constructed by this translation processes, and we compare execution times and numerical errors in discrete time and discrete event simulations of the example models.

2. Discrete event models

To build a discrete event model, we focus on when the next change in the model will occur. This focus is facilitated by dividing the model into components that interact via messages. When working with cellular models, the component is a cell and messages convey information about the cell's neighbors.

Each cell has a state q. This state is influenced by an input x received in messages from the neighboring cells. Likewise, an output y is produced by a cell to inform its neighbors about a change in q. Frequently, a time stepping simulator dispenses with these distinct input and output, instead obtaining the state of a neighboring cell by direct inspection. A useful view of this special circumstance is that input and output are computed by the time stepping simulator via direct inspection. That is, this is a choice of simulator implementation; input and output are nonetheless present in the model.

In the discrete event model, each cell has a time advance $ta(q)$ that is the amount of time the state q will persist if the cell does not receive new input. If t was the time that the cell entered state q, then $t + ta(q)$ is time of next event for that cell. The cells with the smallest time of next event are the imminent components.

The imminent components will change state and report new information, via messages, to their neighbors which, in turn, may change state in response to this input. Hence, the new state of the model as a whole is determined by calculating new states for the imminent components and the components that receive input from an imminent component. The simulation procedure can be summarized as follows [16,17]:

1. Find the smallest time of next event and the components having this time of next event. These are the imminent components.
2. Calculate the output y from the imminent components; this becomes the input x to their neighbors.
3. Calculate the new state of the imminent components and the components receiving input from the imminent components.
4. Calculate the new time of next event for each component that has changed state and repeat from step one.

We build a model of a cell by defining the five elements needed to carry out this simulation procedure:

1. The variables comprising the state q and input x.
2. The output data $y = \lambda(q)$ provided in each state; this becomes the input x received by other components. The function λ is the output function of the model.
3. The time advance $ta(q)$ for each state.
4. How a new state is calculated in the three possible cases; these are
   (a) When the component is imminent. These are called internal events, and the cell's new state q' is given by the internal state transition function $q' = \delta_{\text{int}}(q)$. The time that was spent in state q is $ta(q)$.
   (b) When the component receives input. These are called external events. The cell's response is defined by the external state transition $q' = \delta_{\text{ext}}(q, e, x)$ where e is the time spent in state q and $e \leq ta(q)$.
   (c) When the model is imminent and receives input. These are called confluent events. The cell's response is defined by the confluent transition function $q' = \delta_{\text{con}}(q, x)$. The time that was spent in state q is $ta(q)$.

A trivial transformation from a discrete time to discrete event model keeps the discrete time model intact. In this trivial transformation, the state, input, output, and transition rules are exactly as they appear in the discrete time model. The time advance is fixed at the step size. The result is that every component is imminent and receives input at each time step, and so the evolution of the model is governed by the confluent transition function.

However, this trivial transformation does not take advantage of the computational efficiencies offered by the discrete event simulation procedure. Creativity is indispensable for realizing an efficient discrete event model, but there are several steps typical of any transformation from discrete time model to discrete event model.

1. Given a state q of a component, if the input to this component does not change, then when will it change state? This defines the time advance $ta(q)$. In essence, we are asking when the component will undergo a change of state assuming it is the only active component.
2. At the instant the time advance expires and a change of state is about to occur, what data does the component provide to its neighbor and what is the component's new state? This defines the output $\lambda(q)$ and internal state transition function $\delta_{\text{int}}(q)$.
3. If the component receives input before the time advance expires, then how does the state change? That is, given our present state q, the received input x, and the elapsed time $e \leq ta(s)$, how do we calculate the new state? This defines the external state transition $\delta_{\text{ext}}(q, e, x)$.
4. If the component receives an input x at the same instant the time advance expires, then how does the state change? That is, given the present state q, input x, and that the elapsed time $e = ta(q)$, how do we calculate the new state? This defines the confluent transition function $\delta_{\text{con}}(q, x)$.

Having answered these questions, the resulting computations are encoded using a discrete event simulation package. Typically, these are object oriented libraries, and creating a component involves deriving from a base class that provides virtual (or abstract) methods for realizing the time advance, output function, and the internal, external, and confluent state transitions.

The simulation package provides an efficient implementation of the simulation procedure, which it applies to instances of the derived classes to generate the behavior of the whole model. This is analogous to how many numerical integration packages are used. A system definition is provided in the form of a function that calculates derivatives; this is supplied by the modeler. The simulation package applies a numerical integration procedure to this function to generate behaviors.

3. A probabilistic model of grain growth

We borrow the essential growth mechanism from the cellular automaton proposed in [18]. The state of a cell in this model is its grain orientation C, which is recorded as an integer so that if two cells have the same integer they have the same orientation. The actual angle of orientation is irrelevant to the simulation's progress. At each step of

[1] The software used to perform our simulation experiments is available as the grain growth example that comes with the adevs simulation package at https://sourceforge.net/projects/adevs/.

the cellular automaton, each cell changes its grain orientation with probability $\hat{w}$.

Specifically, for each step, at each cell, we sample a uniform random variable in $[0, 1]$. If this sample is greater than $\hat{w}$, then the grain orientation at that cell is unchanged. Otherwise, a new orientation is determined. Each cell has eight neighbors arranged in a Moore neighborhood. The orientation of neighbor k at step n of the simulation is $C_k(n)$. We select a neighboring cell at random, and the $C_k(n)$ of that neighbor becomes the candidate for the next orientation $C(n+1)$ at the cell undergoing a transition. Whether to accept the new orientation depends on the resulting boundary energy.

At step n, the boundary energy $E(n)$ is equal to the number of neighbors with orientations $C_j(n)$ different from $C(n)$. If $C_k(n)$ is the candidate reorientation, then the boundary energy $E(n+1)$ that would result from this orientation is the number of neighbors such that $C_k(n) \neq C_j(n)$. If $E(n+1) < E(n)$ then $C_k(n)$ becomes the new orientation of the cell; that is, $C(n+1) = C_k(n)$. If $E(n+1) > E(n)$ then the change is rejected and $C(n+1) = C(n)$. Otherwise, with probability $1/2$ we retain $C(n)$ and with probability $1/2$ we accept $C_k(n)$.

For example, suppose $C(n) = 0$ and its eight neighbors have orientations 0, 0, 1, 1, 2, 2, 2, and 3 respectively. The boundary energy at $E(n)$ is 6; two neighbors have orientation zero and six have orientations that are not zero. If we select 3 as our candidate orientation for step $n+1$, then $E(n+1) = 7$. In this case, the change in orientation is rejected.

To obtain a discrete event version of this model, we follow the procedure sketched in Section 2. The state q comprises the orientation C of the cell, the reported orientations C_1, ..., C_8 of its neighbors, and a new variable h that is the time remaining until the cell could change orientation given the current C, C_1, ..., C_8. The output function λ is the new orientation C' that the cell will obtain. This is calculated using the procedure described above to minimize energy. The time advance simply reports h. Hence,

$$ta((C, C_1, \ldots, C_8, h)) = h \text{ and}$$
$$\lambda((C, C_1, \ldots, C_8, h)) = C' .$$

The internal transition function replaces C with the new orientation C' and calculates a new value h' for the time until the next internal event. The external transition function reduces h by e and replaces the prior neighbor orientation C_j with the newly received orientation C'_j. The confluent transition function is defined by composing the internal and external transition functions, hence

$$\delta_{\text{int}}((C, C_1, \ldots, C_8, h)) = (C', C_1, \ldots, C_8, h'),$$
$$\delta_{\text{ext}}((C, C_1, \ldots, C_8, h), e, C'_j) = (C, C_1, \ldots, C'_j, \ldots, C_8, h - e), \text{and}$$
$$\delta_{\text{con}}((C, C_1, \ldots, C_8, h), C'_j) = \delta_{\text{ext}}(\delta_{\text{int}}((C, C_1, \ldots, C_8, h)), 0, C'_j).$$

It remains to show how h is calculated. If the orientations are such that a change of orientation is not possible, then $h = \infty$. Such a cell is idle. Otherwise, h is selected by sampling an exponential random variable with the rate parameter w/s. We turn now to the derivation of this rate parameter.

3.1. From Bernoulli to Poisson

To begin, we review the role that transition probability $\hat{w}$ in the cellular automaton plays in approximating a Poisson process, and how it converges to a Poisson process as the time represented by a step of the automaton becomes very small. To arrive at this conclusion we begin with the method put forward by Raabe [1,2] to derive $\hat{w}$. This parameter originates from two factors. The first is a physically motivated rate of advance w, which is derived from Turnbull's rate expression. This physically motivated parameter is further adjusted by the scaling factor in space s and a grid attack frequency v_0, which determine the resolution of the model in space and time respectively. These choices by the modeler determine the probability $\hat{w} = w/(sv_0)$, and each simulation step represents $h = 1/v_0$ units of time.

In a unit interval of time, the cellular automaton takes v_0 steps. The likelihood of a transition at each step is identical to and independent from the likelihood at any other step. Therefore, the cellular automaton regulates the rate of transitions in such a way that the likelihood of obtaining k transitions in a unit of time is

$$P(k) = \frac{v_0!}{k!(v_0 - k)!}\left(\frac{w}{v_0 s}\right)^k \left(1 - \frac{w}{v_0 s}\right)^{v_0 - k} . \tag{1}$$

It is well known (see, e.g., [19]) that as v_0 grows to infinity the probability $P(k)$ converges to

$$\lim_{v_0 \to \infty} P(k) = \frac{1}{k!}\left(\frac{w}{s}\right)^k \exp(-w/s) . \tag{2}$$

which is characteristic of a Poisson process with rate w/s. It follows that as $v_0 \to \infty$ the interval h between state transitions is described by the probability density function (again see, e.g., [19])

$$f(h; w/s) = (w/s)\exp(-hw/s) . \tag{3}$$

Consequently, the discrete event model, by using a real valued time advance, does not require the modeler to select v_0. Instead, using $f(h; w/\lambda)$ and a single draw p from a uniform random variable in $[0, 1]$, the time to the next reorientation at the cell is

$$h = -\frac{1}{w/s} \ln \frac{p}{w/s} . \tag{4}$$

3.2. Bernoulli errors

Turnbull's rate expression describes a process that is continuous in time and properly expressed as a Poisson process. For this reason, the discrete event model is exact regarding when transitions should occur. Relative to this exact discrete event model, a discrete time model with its fixed rate v_0 contains a numerical error. In a time stepping simulation, the probability of obtaining a change of state at the nth step is

$$\hat{w}(1 - \hat{w})^{n-1} = \frac{w}{sv_0}\left(1 - \frac{w}{v_0 s}\right)^{n-1} . \tag{5}$$

However, the exact Poisson process gives a different probability of the next state transition occurring after n steps. The nth step corresponds to an interval of time n/v_0, and so the cumulative probability of the Poisson process is

$$\int_0^{n/v_0} (w/s)\exp(-hw/s)\, dh = (w/s)\left(1 - \exp\frac{-nw/s}{v_0}\right) . \tag{6}$$

The difference between the Bernoulli probability in Eq. (5) and the Poisson probability in Eq. (6) is

$$\frac{w}{sv_0}\left(1 - \frac{w}{sv_0}\right)^{n-1} - (w/s)\left(1 - \exp\frac{-nw/s}{v_0}\right) . \tag{7}$$

This is the error incurred by the discrete time simulation relative to the more precise discrete event simulation.

For the purpose of comparing the two simulation procedures, it is convenient to use the same rate parameter in the discrete event and discrete time simulation. Unconcerned with the physical interpretation of our example, let us set the product $sv_0 = 1$ so that $w = \hat{w}$. If w is small, then by using w as the rate parameter in our discrete event simulation we obtain a result close to the discrete time simulation. The time advance should now be interpreted as (fractional) steps of the time stepping simulator, and the difference in transition probabilities at n steps is

$$w(1 - w)^{n-1} - w\left(1 - \exp(-wn)\right) . \tag{8}$$

This expression for the error is derived from Eq. (7) when the model's rate parameter is fixed at w.

A plot of the error given by Eq. (8) is shown in Fig. 1 for several choices of w. The probability of an early transition in the discrete

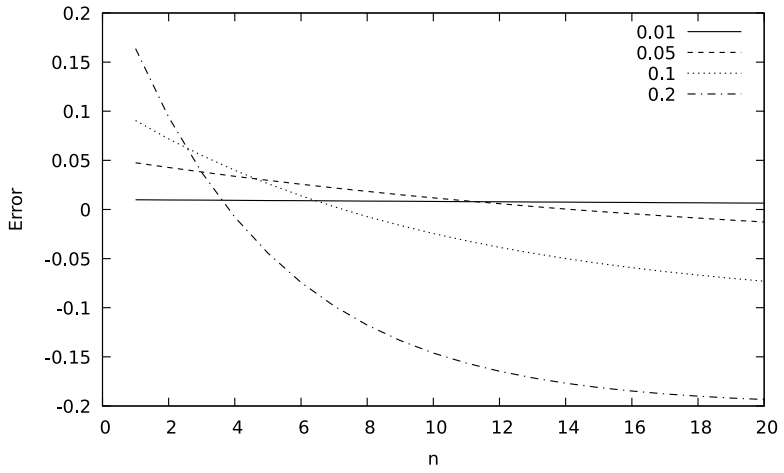


Fig. 1. Errors indicated by Eq. (8) for $w = 0.2$, 0.1, 0.05, and 0.01.

time simulation is large relative to the exact discrete event simulation, suggesting the error creates a faster rate of evolution than should occur. Of course, this error diminishes with w indicating that a slower process is simulated with less error than a quicker process if the resolution in time and space is fixed (in this example, fixed so that $sv_0 = 1$).

We have assumed for convenience that $w < 1$ when picking the resolution of the model such that $\hat{w} = w$. More generally, $\hat{w} < 1$ is achieved by picking a suitably refined resolution in time and space. Referring back to Eq. (7), we note that the error with regard to growing v_0 is the same one shown in Fig. 1. That is, when using a time stepped simulation procedure that implements a Bernoulli process, finer resolution in time leads to smaller errors. The discrete event simulation avoids error by sampling directly from the exponentially distributed intervals of the Poisson process.

3.3. Computational efficiency

The time required to perform a simulation with the discrete time simulator is simple to calculate. Assume, on average, that to calculate a single state transition at a cell requires one unit of time. To simulate a two dimensional model with $N \times N$ cells for T steps requires N^2T units of time.

A state transition at a cell in the discrete event simulation carries with it the computational overhead of sorting cells by their time of next event, the test for quiescence, and the cost of initializing the cell at simulation start. In any given simulation execution, this overhead will tend to be a constant so that the cost of a state change relative to the discrete time simulation is some $a > 1$. The value of a will depend on the scheduling algorithm used by the discrete event simulator and the number of cells in the model.

The number of state transitions calculated in an interval of time will depend on w and the fraction of the cell space that is quiescent. On average, an active cell in the discrete event model will calculate w state transitions for each state transition of a cell in the cellular automaton. If the fraction of the space that is active is α then the total cost of simulating for T steps is

$$\alpha N^2 T w a \ . \tag{9}$$

The speedup of the discrete event simulation relative to the discrete time simulation will be

$$\frac{N^2 T}{\alpha N^2 T w a} = \frac{1}{\alpha w a} \ . \tag{10}$$

In a model of grain growth, α will be related to the grain size. If grains are approximately circular with radius r and if only the cells on a grain boundary are active, then α will be close the ratio of the area to the perimeter

$$\alpha \approx 2\pi r / r^2 = 2\pi / r \ . \tag{11}$$

The grains grow in size as time progresses, and so it is the average grain size over the interval T that determines the speedup. Writing this as $\hat{r}(T)$, the speedup is

$$\frac{\hat{r}(T)}{2\pi w a} \tag{12}$$

We will not explore the three dimensional case here except to note that Eq. (12) will change only by multiplication with a constant. The fraction α becomes the ratio of surface area to volume, which is proportional to $r^2/r^3 = 1/r$.

To demonstrate this speedup law, the discrete time and discrete event simulations were implemented in C++. Total execution time and total transition counts for several choices of w, N, and T are listed in Table A.1 (see the appendix). These model parameters are selected so that each simulation covers that same span of time if we assume a fixed physical dimension for the space spanned by the model. The Linux clock function is used to measure time. The model is initialized with a unique orientation at each cell.

The overhead a imposed by the discrete event simulation algorithm relative to the discrete time simulation is measured by counting the total number of state transitions in each simulation, dividing these by the total execution time, and then normalizing. These measurements are shown in Fig. 2 for the simulation experiments tabulated in Table A.1. The essentially flat overhead curves are a product of the sorting algorithm used by the simulation package.

This sorting algorithm that determines a imposes an overhead proportional the $\log_2 M$, where M is the number of active cells. In the worst case, $M = N^2$, but generally M is much smaller than this because of the quiescent cells. Nonetheless, increasing N leads to greater overhead at a rate similar to $2\log_2 N$. The more slowly evolving grains created by reducing w results in fewer quiescent cells and so a larger M and somewhat larger overhead. The slight decline as a function of execution time seen in Fig. 2 is due to an increase in the number of idle cells as the simulation progresses.

Fig. 3(a) shows the change in $\hat{r}(T)$ as estimated using Eq. (12) from the known w and T, the measured speedup, and a. Fig. 4 emphasizes the relationship between average grain size, time, and speedup. There, the grain patterns grown by the cellular automaton and discrete event model at $T = 1500$ and $T = 48{,}000$ are shown for $N = 300$ and $w = 0.01$. The growth in grain size is readily apparent in the sequence of plots, agreeing qualitatively with the speedup seen in Fig. 3(b) and estimated

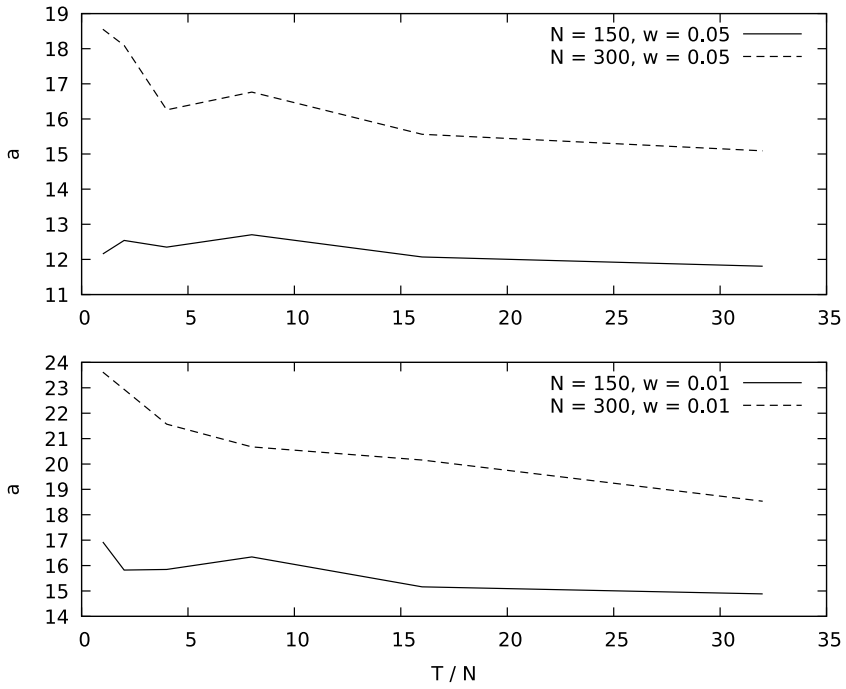


Fig. 2. State transition overhead of the discrete event simulation relative to the discrete time simulation for each of the scenarios presented in Table A.1.

grain size in Fig. 3(a). The estimated grain size grows approximately in proportion to $\sqrt{t}$, which is the expected shape of the growth rate for an idealized physical model [20]. This is illustrated for two cases where the constant of proportionality was chosen so that the idealized rate and simulation rate are close.

4. De-centered square for grain growth

The decentered square (or, in 3D, octahedron) cellular automaton (DoCA) was first introduced in 2D [21] and then 3D [22] to model grain growth during solidification. The DoCA is typically implemented with a time stepped evolution of the size of the grain envelop until the center of a neighboring cell is encompassed by the growing envelope. Once this happens, the neighboring cell is considered captured and then proceeds to propagate its own grain envelope.

The time step has historically been selected to having a grain envelope grow in size by at most 1/25 of a cell length at each step [22], although larger time steps, such as 1/5 of a cell at each step [23], have also been used to improve computational speed. Because a cell can only be captured once, having multiple capture events at a cell within the same time step can lead to an incorrect assignment. This type of error occurs more frequently as the step size is increased. Just as in the previous example, selecting a step size requires a trade between execution time and accuracy.

Our second example of a discrete event simulation concerns the 2D DoCA with a constant growth rate. Though DoCA is used to model grain growth in real materials, this example does not consider any specific material and thus both the growth rate and nucleation density are non-dimensional. The growth rate is the fraction of a cell length an envelope is allowed to grow per step. The nucleation density is the fraction of the total number of cells from which a grain can originate. To arrive at a discrete event simulation, the state of a cell comprises the state variables in the discrete time simulation: angle of orientation, size of the grain, center of the grain, if the cell has been captured, and its capture status (true or false). We add to these state variables the time h remaining to the next capture event and the neighbor that will be captured at that time. The cell's time advance reports h and the output function produces a message for the neighboring cell that will be captured.

At an internal event, the cell expands its grain envelope and calculates the time and location of the next capture event. If all of the neighboring cells have been captured, then $h = \infty$. Otherwise, $h = d/v$ where d is the distance the diagonals of the grain envelope need to grow for the grain envelope to reach the center of the nearest uncaptured cell and v is the rate of growth of these diagonals. At an external event, the cell becomes captured if it is not already captured. If it becomes captured, h is set as just described. Otherwise, h remains at ∞ or is set to $h - e$. As before, the confluent state transition function is the composition of the internal and external state transition functions.

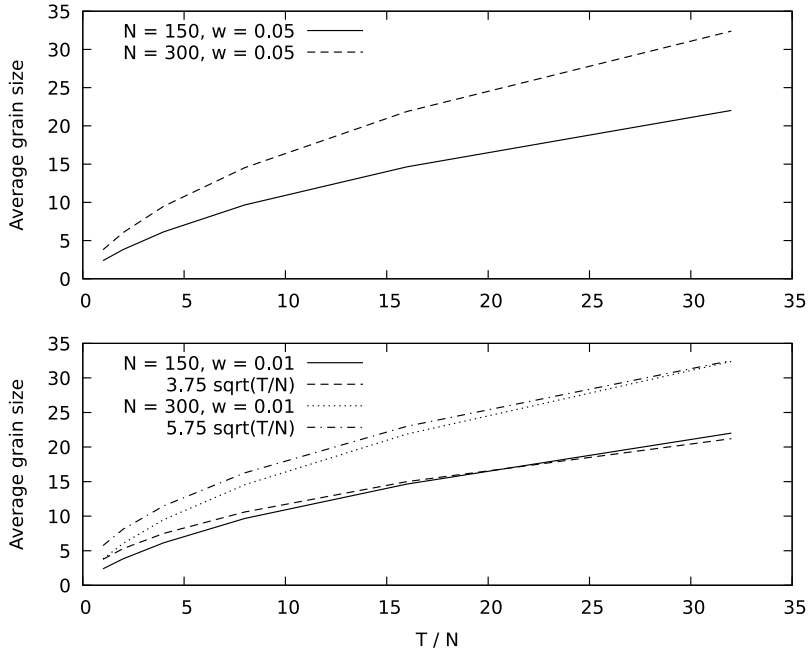
The model is initialized by assigning a nucleation time to a fraction of the space. These cells capture themselves spontaneously after a fixed interval of time if they are not captured by a grain before that interval expires. The orientation of these nucleating sites is selected at random.

The speedup of the discrete time simulation relative to the discrete event simulation depends chiefly on the step size used by the discrete time simulator. For a simulation of N cells over an interval T using a step size Δt, the discrete time simulator requires $NT/\Delta t$ state transitions to be calculated. The discrete event model has a fixed upper limit of 17 state transitions at each cell. Hence, for N cells we require at most $M < 17N$ state transitions. If $a > 1$ is the overhead imposed by the scheduler in the discrete event simulation, the speedup of the discrete event simulator relative to the discrete time simulator is

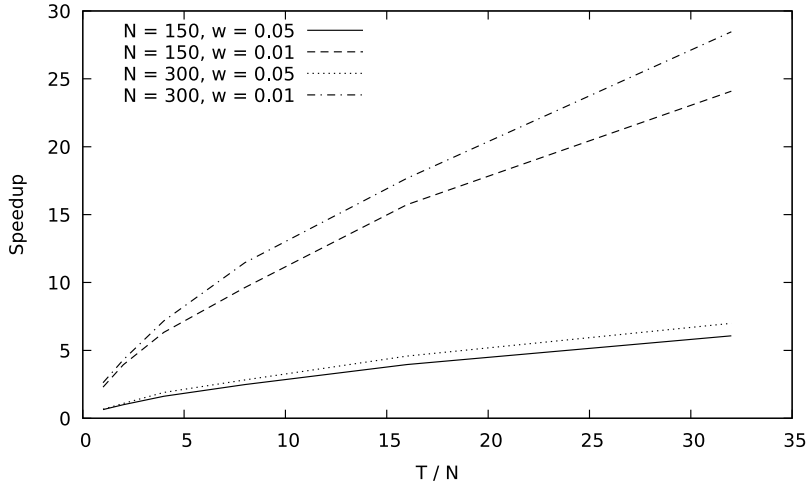
$$\frac{NT}{\Delta t}\frac{1}{MNa} = \frac{T}{M\Delta t a} = \frac{k}{\Delta t} \tag{13}$$

where $k = T/(Ma)$.

This speedup curve is shown in Fig. 5. For this speedup experiment the model is simulated until each cell has been captured. The fraction of the space that nucleates spontaneously affects the constant k by changing the simulation interval T and the number of events M calculated at each cell. Nonetheless, the primary dependence of the speedup on $1/\Delta t$ is apparent in the plot. The choice of Δt has a practical lower limit of 1/100 in this example. Fig. 6 shows this solution, and solutions calculated with a time step below this lower limit are indistinguishable because each cell experiences a single capture event. The speedup in this case is 20x relative to the discrete time simulation.



(a) Grain size estimated from speedup measurements, measurement of a, known w and T, and Eqn. 12.



(b) Measured speedup of discrete event simulation relative to the discrete time simulation

Fig. 3. Speedup and grain size in the discrete event simulation.

5. Conclusion

The two example models presented here highlight the key benefits of applying discrete event simulation to cellular automata models of grain growth. The discrete event algorithms achieve speedup by focusing computational effort on the active cells in the model and eliminate errors by advancing time precisely to the events of interest. In the probabilistic model, this avoids error inherent in approximating the underlying Poisson process. The discrete event formulation of the DoCA model ensures that exactly one capture event happens at each cell.

For the sake of clarity, these simple examples omit physics that could complicate the time advance function. An important applications of the DoCA model involves coupling to a finite element model (FEM) of a thermodynamic system [3]. One-way coupling consists of sending local thermal information from the FEM to the DoCA model, whereas two-way coupling consists of then having the DoCA model send back information about how much material has solidified. In these coupled problems, the rate of growth of the grain boundary changes with time in response to the thermodynamic system, making the simple projection used in our example impractical.

Solutions for these more practical and more complicated problems is a topic for future research. In some cases, it may be sufficient to apply known techniques for the numerical solution of (partial) differential equations within a discrete event simulation [16,24,25]. In other cases, the development of new models or techniques may be necessary. For example, using only the time and cooling rate of a melting event, it may be possible to solve for the exact capture time if a suitable interface response function is chosen. A low order polynomial fit could be solved directly for the time of the next capture event. Using a power law such

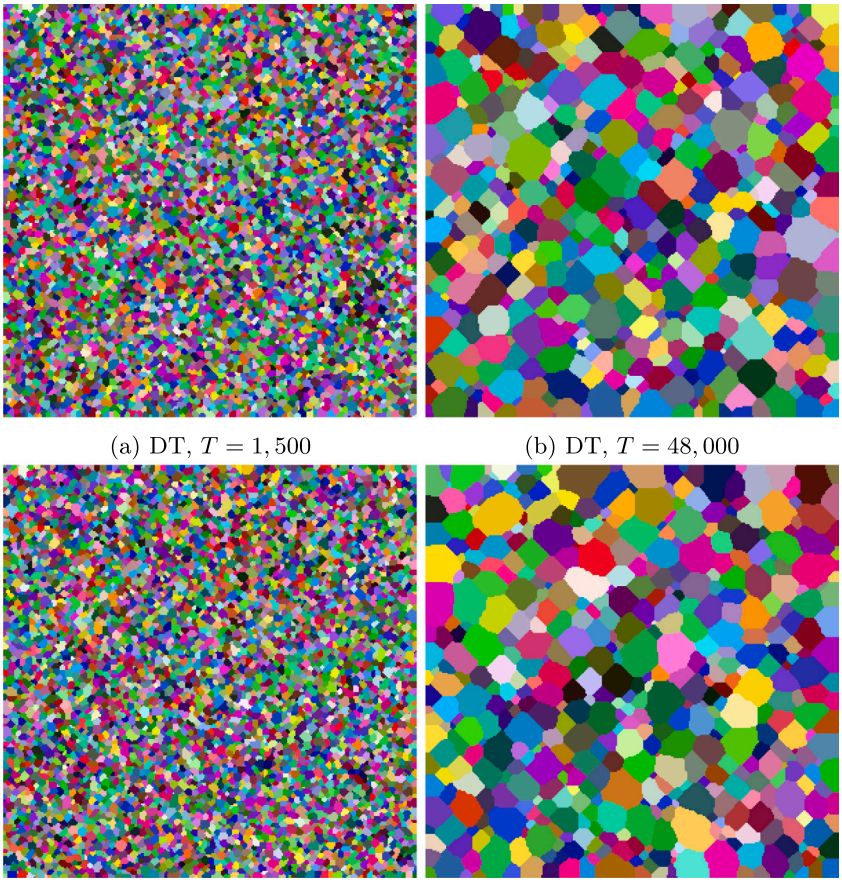


Fig. 4. Grains grown in the discrete event (DE) and discrete time (DT) simulations. The cell space is 300 × 300.

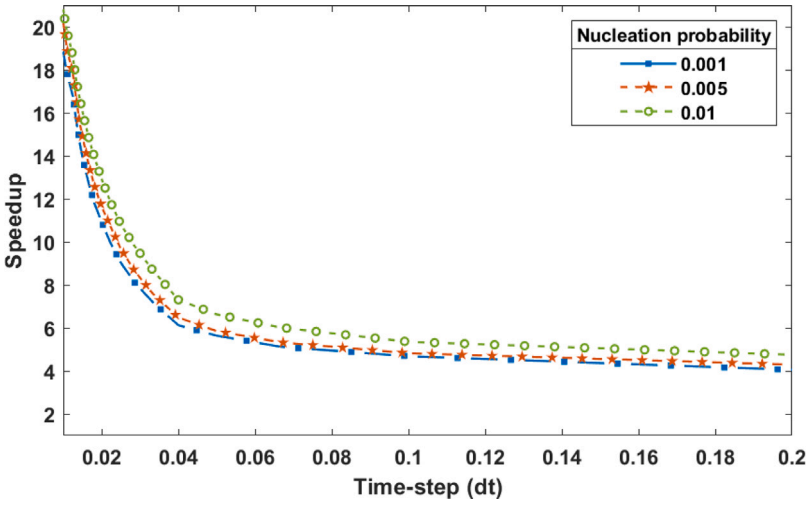


Fig. 5. Speedup of discrete event model relative to the cellular automaton.

as $V(\Delta T) = A(\Delta T)^B$ for the interface response function [26] also makes calculating the time of capture events straightforward.

With the surge in popularity of Additive Manufacturing (AM), DoCA models have seen heavy use in large-scale simulations. Since many AM processes have large thermal gradients, large cooling rates, and a small fraction of the domain in liquid form at any given time, simulation of any kind is typically costly. The most common approach involves one-way coupling of the DoCA model to a model for heat transport, although these simulations have taken days to run [27–29]. Recently a different approach, which does not involve coupling of the numerical solvers, uses a reduced data format that encodes only vital information about solidification, and these simplified models have been successfully

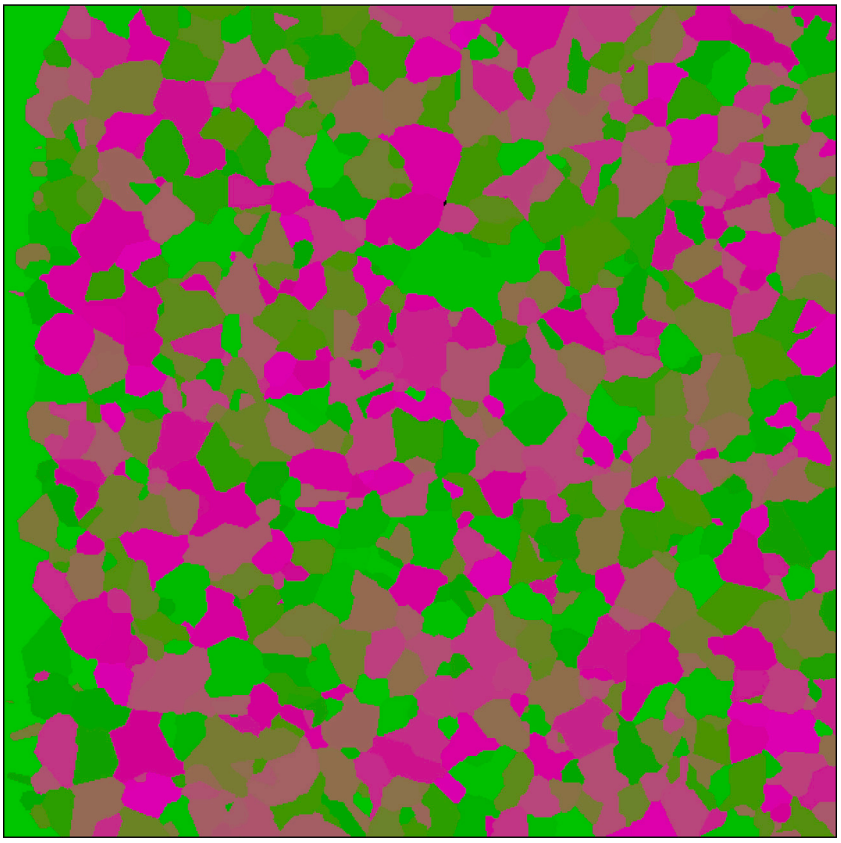


Fig. 6. DoCA algorithm with time step 1/100. The cell space is 600 × 600.

Table A.1
Simulation running times and state transition counts for the selected model configurations. Speedup is discrete time simulation (DT) running time/discrete event simulation (DE) running time.

w	N	T	DT time	DT counts	DE time	DE counts	Speedup
0.05	150	150	72671	3375000	113703	434428	0.639
0.05	150	300	139537	6750000	140223	540889	0.995
0.05	150	600	273839	13500000	168674	673318	1.62
0.05	150	1200	537476	27000000	216937	857985	2.48
0.05	150	2400	1063516	54000000	268989	1131750	3.95
0.05	150	4800	2144793	108000000	353048	1505820	6.08
0.01	150	750	274532	16875000	119653	434428	2.29
0.01	150	1500	548217	33750000	139030	540889	3.94
0.01	150	3000	1066221	67500000	168542	673318	6.33
0.01	150	6000	2132721	135000000	221465	857985	9.63
0.01	150	12000	4255555	270000000	270455	1131750	15.7
0.01	150	24000	8513285	540000000	353436	1505820	24.1
0.05	300	300	531174	27000000	796988	2183600	0.666
0.05	300	600	1039835	54000000	952761	2734736	1.09
0.05	300	1200	2247851	108000000	1181496	3491166	1.90
0.05	300	2400	4353459	216000000	1540635	4559428	2.83
0.05	300	4800	8475025	432000000	1850272	6060432	4.58
0.05	300	9600	16836433	864000000	2408221	8188391	6.99
0.01	300	1500	2088987	135000000	798049	2183600	2.62
0.01	300	3000	4175458	270000000	970023	2734736	4.30
0.01	300	6000	8381972	540000000	1168745	3491166	7.17
0.01	300	12000	16659350	1080000000	1453958	4559428	11.5
0.01	300	24000	33347265	2160000000	1886000	6060432	17.7
0.01	300	48000	67969468	4320000000	2387508	8188391	28.5

used to predict grain structure [30]. More recent work using a reduced format [31] has shown improved running times through the use of GPU enabled code; however, because only a small fraction of the domain is liquid at any time, it still scales poorly on multi-node computing architectures due to most work being done on a single node.

Parallel algorithms for discrete event simulation may offer new opportunities for using high performance computers to simulate complex DoCA models. Introductions to these parallel algorithms can be found in [16,17,32] and a recent, preliminary application of these parallel algorithms to kinetic Monte Carlo problems is described by Oppelstrup

et al. [33]. The excellent performance of these algorithms on very large scale parallel computers derives from their elimination of barrier synchronization at the time step. Each part of the parallel execution proceeds independently until an exchange of information becomes necessary. In a well constructed parallel discrete event simulation, this operating principle is used to advance each processor to a point in time where its assigned portion of the model is active, thereby avoiding idle processors and achieving excellent speedup.

CRediT authorship contribution statement

James Nutaro: Constructed the discrete event simulations, and produced the analytical results. **Benjamin Stump:** Constructed the discrete time DoCA simulation and contributed to the manuscript. **Pratishtha Shukla:** Performed the speedup experiments and contributed to the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Appendix

Table A.1 lists state transition counts and execution times observed in the speedup experiments that we describe in Section 3.3.

References

[1] D. Raabe, Introduction of a scalable three-dimensional cellular automaton with a probabilistic switching rule for the discrete mesoscale simulation of recrystallization phenomena, Phil. Mag. A 79 (10) (1999) 2339–2358.

[2] D. Raabe, Cellular automata in materials science with particular reference to recrystallization simulation, Annu. Rev. Mater. Res. 32 (1) (2002) 53–76.

[3] C. Gandin, J.-L. Desbiolles, M. Rappaz, P. Thevoz, et al., A three-dimensional cellular automation-finite element model for the prediction of solidification grain structures, Metall. Mater. Trans. A 30 (12) (1999) 3153–3165.

[4] C.-A. Gandin, M. Rappaz, A coupled finite element-cellular automaton model for the prediction of dendritic grain structures in solidification processes, Acta Metall. Mater. 42 (7) (1994) 2233–2246, http://dx.doi.org/10.1016/0956-7151(94)90302-6, URL https://www.sciencedirect.com/science/article/pii/0956715194903026.

[5] D. Svyetlichnyy, Modelling of the microstructure: From classical cellular automata approach to the frontal one, Comput. Mater. Sci. 50 (1) (2010) 92–97, http://dx.doi.org/10.1016/j.commatsci.2010.07.011, URL https://www.sciencedirect.com/science/article/pii/S0927025610004210.

[6] C. M., S. M., M. L., The role of neighborhood density in the random cellular automata model of grain growth, Comput. Methods Mater. Sci. 21 (3) (2021) 129–137, http://dx.doi.org/10.7494/cmms.2021.3.0760.

[7] Y. Lian, S. Lin, W. Yan, W.K. Liu, G.J. Wagner, A parallelized three-dimensional cellular automaton model for grain growth during additive manufacturing, Comput. Mech. 61 (2018) 543–558.

[8] A. Muzy, B.P. Zeigler, Introduction to the activity tracking paradigm in component-based simulation, Open Cybern. Syst. J. 2 (2008) 30–38.

[9] A. Muzy, R. Jammalamadaka, B.P. Zeigler, J.J. Nutaro, The activity-tracking paradigm in discrete-event modeling and simulation: The case of spatially continuous distributed systems, Simulation 87 (5) (2011) 449–464.

[10] B. Zeigler, Discrete event models for cell space simulation, Internat. J. Theoret. Phys. 21 (1982) 573–588.

[11] G.A. Wainer, The Cell–DEVS formalism as a method for activity tracking in spatial modelling and simulation, Int. J. Simul. Process Model. 10 (1) (2015) 19–38.

[12] G.A. Wainer, N. Giambiasi, Application of the Cell-DEVS paradigm for cell spaces modeling and simulation, Simulation 71 (1) (2001) 22–39, URL http://cell-devs.sce.carleton.ca/publications/2001/WG01.

[13] G.A. Wainer, Cellular modeling with Cell–DEVS: A discrete-event cellular automata formalism, in: J. Wąs, G.C. Sirakoulis, S. Bandini (Eds.), Cellular Automata, Springer International Publishing, Cham, 2014, pp. 6–15.

[14] H. Vangheluwe, G.C. Vansteenkiste, The cellular automata formalism and its relationship to DEVS, in: Proceedings of the 14th European Simulation Multiconference on Simulation and Modelling: Enablers for a Better Quality of Life, SCS Europe, 2000, pp. 800–810.

[15] G.A. Wainer, Discrete-Event Modeling and Simulation: A Practitioner's Approach, first ed., CRC Press, Inc., 2009.

[16] B.P. Zeigler, A. Muzy, E. Kofman, Theory of Modeling and Simulation, 3rd Edition, Elsevier, 2018.

[17] J. Nutaro, Building Software for Simulation, Wiley, 2010.

[18] S. Liua, Y. Jianga, R. Lua, X. Cheng, J. Lia, Y. Chen, G. Tian, Cellular automata simulation of grain growth of powder metallurgy nickel-based superalloy, 2021, arXiv preprint arXiv:2106.04888.

[19] A. Papoulis, S.U. Pillai, Probability, Random Variables and Stochastic Processes, 4th Edition, McGraw-Hill Europe, 2002.

[20] H. Atkinson, Overview no. 65: Theories of normal grain growth in pure single phase systems, Acta Metall. 36 (3) (1988) 469–491, http://dx.doi.org/10.1016/0001-6160(88)90079-X, URL https://www.sciencedirect.com/science/article/pii/000161608890079X.

[21] M. Rappaz, C.-A. Gandin, Probabilistic modelling of microstructure formation in solidification processes, Acta Metall. Mater. 41 (2) (1993) 345–360.

[22] C.-A. Gandin, M. Rappaz, A 3D cellular automaton algorithm for the prediction of dendritic grain growth, Acta Mater. 45 (5) (1997) 2187–2195.

[23] Y. Lian, S. Lin, W. Yan, W.K. Liu, G.J. Wagner, A parallelized three-dimensional cellular automaton model for grain growth during additive manufacturing, Comput. Mech. 61 (5) (2018) 543–558.

[24] J. Nutaro, Discrete-event simulation of continuous systems, in: P.A. Fishwick (Ed.), Handbook of Dynamic System Modeling, Chapman and Hall/CRC, 2007.

[25] F.E. Cellier, E. Kofman, Continuous System Simulation, Springer, 2006.

[26] O. Zinovieva, A. Zinoviev, V. Ploshikhin, Three-dimensional modeling of the microstructure evolution during metal additive manufacturing, Comput. Mater. Sci. 141 (2018) 207–220.

[27] T. Carozzani, C.-A. Gandin, H. Digonnet, Optimized parallel computing for cellular automaton–finite element modeling of solidification grain structures, Modelling Simul. Mater. Sci. Eng. 22 (1) (2013) 015012.

[28] K. Teferra, D.J. Rowenhorst, Optimizing the cellular automata finite element model for additive manufacturing to simulate large microstructures, Acta Mater. 213 (2021) 116930.

[29] D. Kats, Z. Wang, Z. Gan, W.K. Liu, G.J. Wagner, Y. Lian, A physics-informed machine learning method for predicting grain structure characteristics in directed energy deposition, Comput. Mater. Sci. 202 (2022) 110958.

[30] M. Rolchigo, B. Stump, J. Belak, A. Plotkowski, Sparse thermal data for cellular automata modeling of grain structure in additive manufacturing, Modelling Simul. Mater. Sci. Eng. 28 (6) (2020) 065003.

[31] M. Rolchigo, S.T. Reeve, B. Stump, G.L. Knapp, J. Coleman, A. Plotkowski, J. Belak, ExaCA: A performance portable exascale cellular automata application for alloy solidification modeling, Comput. Mater. Sci. 214 (2022) 111692.

[32] R.M. Fujimoto, Parallel and Distributed Simulation Systems, Wiley, 2000.

[33] T. Oppelstrup, D.R. Jefferson, V.V. Bulatov, L.A. Zepeda-Ruiz, SPOCK: Exact parallel kinetic Monte-Carlo on 1.5 million tasks, in: Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, in: SIGSIM-PADS '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 127–130, http://dx.doi.org/10.1145/2901378.2901403.