Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

PERFORMANCE AND SOUNDNESS OF SIMULATION: A CASE STUDY BASED ON A CELLULAR AUTOMATON FOR IN-BODY SPREAD OF HIV

Till Köster
Institute for Visual and Analytic Computing
University of Rostock
Germany

Philippe J. Giabbanelli
Dept. of Computer Science & Software Engineering
Miami University, Oxford, OH 45056
USA

Adelinde Uhrmacher
Institute for Visual and Analytic Computing
University of Rostock
Germany

ABSTRACT

Cellular automata are often used for spatial dynamics in cell biology such as the spread of infections within a cell biological population. Due to the number of cells in a model, efficient simulation algorithms have received increasing attention over the last decade. Many cellular automata models are executed synchronously. In this paper, we focus on improving the efficiency of a cellular automaton model known as the 'dos Santos' model for the Human Immunodeficiency Virus (HIV). The synchronous updates in this model are driven by mostly deterministic transitions and a few highly probabilistic transitions. This design can be exploited to create an efficient simulation algorithm. We propose such an algorithm using an advanced scheme to generate random numbers (to efficiently perform probabilistic transitions) that efficiently leverages CPU vectorization. The benchmarks show a significant performance increase by a factor of 2.4x in comparison to a quality baseline implementation. We note several limitations in the model behavior and simulation outputs, which apply not only to the dos Santos design but also to the many synchronous HIV models inheriting its design. Several mitigation schemes are discussed to address some of these limitations.

1 Introduction

Over 30 million people have died from complications associated with HIV and AIDS. By the end of 2018, it is estimated that between 32.7 and 44.0 million were living with HIV (World Health Organization 2020). Modeling & Simulation (M&S) has been used in HIV research for several decades to understand the spread of HIV (both within the body and across individuals) or evaluate the potential outcomes associated with interventions. M&S is particularly useful as a first step in intervention design as it allows to predict effects in a 'virtual laboratory' without causing harm to real-world individuals. Two successful modeling projects include Kaplan's early work, which contributed to distributing clean needles to injection drug users (Kaplan 1989), and the study directed by Granich, which strengthened the evidence base for universal testing and immediate therapy (Granich et al. 2009).

Cellular Automata (CA) provide one of the approaches used to simulate HIV. This approach is most commonly used to model the spread within the body, as the regular packing of cells within CA provides an analogy to the tight packing of immune cells in lymphoid tissues which are targeted by the virus. Although several CA models of the immune response to the virus were proposed in the early 1990s (Pandey and Stauffer 1990; Kougias and Schulte 1990), this approach gained attention with the model in (dos Santos and Coutinho 2001), now commonly called the 'dos Santos model'. Its popularity is partly due to its ability to replicate the phases of HIV infection (i.e. before the onset of AIDS) despite its apparent simplicity.

978-1-7281-9499-8/20/$31.00 ©2020 IEEE

Since many stochastic in-body HIV CA models extend the structure of the dos Santos model (Precharattana 2016), investigating its performance and soundness allows to examine the broader field of HIV CA. As the simple implementations prevalent in the field are very time intensive to run, simulations of these stochastic models often use too few runs to yield tight confidence intervals, hence results may not be entirely replicated in future studies (Giabbanelli et al. 2019). Although these models succeed at producing realistic average curves for viral load, we may be *obtaining the right results for the wrong reasons* since the models are driven by very high rates of cell-to-cell infection (i.e. between close neighbors) unlike the real-world mix of cell-to-cell *and* cell-free infections via the bloodstream (Giabbanelli et al. 2019). Recent studies demonstrate performances of the dos Santos model can be improved through just in time compilation of Python code (Giabbanelli et al. 2020) or by using machine learning meta-models (Fisher et al. 2020). To further improve upon these results, we need to study the model's performance characteristics.

Our study builds on previous work in two ways:

1. We increase performance by a factor of 2.4 by performing the first in-depth performance study for the dos Santos CA model and using algorithmic optimizations in a modern native language (Rust).
2. We examine whether the design of CA models (particularly the synchronous transitions and their high probabilities) produces behavior that differs from real-world expectations.

The paper is structured into four parts. In Section 2, we provide a brief background on the design and implementation of the biological CA models, including the dos Santos model. In Section 3, we introduce our approach to profile an optimization. Based on our analysis of the technical properties of the implementation, we propose algorithmic performance improvements in section 4. Finally, we reflect on the soundness of the overall approach.

2 Background: Design and Optimization of CA Models in Cell Biology

A cellular automaton is a discrete model in three ways: cells are updated at discrete time steps, the space is discretized into a regular tiling (such as a grid or a hexagonal tiling), and cells are in discrete states. The rules to update the cells can involve probabilities (hence creating a *stochastic* model), time (e.g., infected cells will die at the next step), or the states of neighbors (e.g. being next to an infected cell makes it possible to pass on virions). Models without probabilistic transitions are *deterministic*, while those with at least one such transition are *stochastic*. All CA for HIV are *synchronous*, which means that cells are all updated at the same time by using a copy to store their future states before committing them.

CA used to model other biological systems, such as cancer growth, typically use *asynchronous* synchronization (Deutsch et al. 2005; Metzcar et al. 2019). This introduces new possibilities for optimization, such as finding efficient means to iterate through the cells or selecting a random neighbor (Poleszczuk and Enderling 2014). A common approach to improving performance for cellular automata is to leverage their 'embarrassingly parallel' update step. Parallelization has been done many times in CA (Salguero et al. 2019), including by running them over GPU-based implementations (Rybacki et al. 2009). Other improvements have leveraged the trade-off between runtime performance and memory footprint, for instance by using memoization algorithms that exploit spatial regularities (Gosper 1984; Caux et al. 2010).

The dos Santos model and its successors stand out among biological models for three defining characteristics: transitions are mostly step-based, very high probabilities are used in stochastic transitions, and only the transition from 'healthy' (or variations thereof) to 'infected' involves neighbors. The structure of the dos Santos model (Figure 1) consists of four main states (healthy, infected A_1, infected A_2, dead) where deterministic transitions occur either at constant time intervals (e.g., A_1 turns into A_2 after 4 steps, A_2 turns into dead in the next step) or are driven by very high probabilities (e.g., dead cells are replenished by the system with 99% probability). The dos Santos model has inspired many derived models, which often consist of adding states and accompanying transitions to explore key concepts such as viral reservoirs (Shi et al. 2008), therapy (González et al. 2013), or adherence to treatment (Rana et al. 2015).

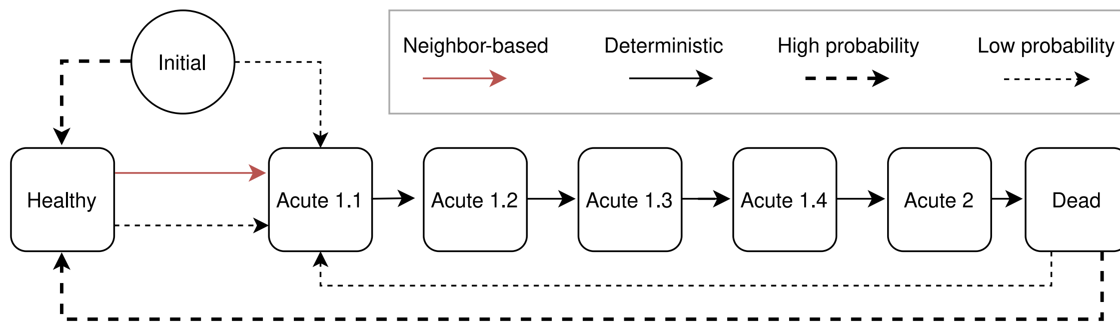

Figure 1: The dos Santos model has 4 states (healthy, infected A_1 and A_2, dead). As A_1 turns A_2 over four time steps, we break it down into four consecutive states, resulting in 7 states. Note that most transitions are either deterministic (plain line) or have a very high probability (broad dashed line).

This 'line of models' perpetuates several design *choices* such as step-based transitions and tends to be faithful to the initial model even in the parameter values. For instance, the dos Santos model uses 4 steps to go from A_1 to A_2. This was still the case for models published up to 15 years later (González et al. 2013; Rana et al. 2015; Precharattana et al. 2010; Moonchai and Lenbury 2016). Similarly, the 99% probability of replacing a dead cell by a healthy one was still in effect 16 years later (Golpayegani et al. 2017).

Although some HIV CA models are occasionally created entirely from scratch and hence bear no resemble to the dos Santos model (Hillmann et al. 2017), the fact that a robust line of research inherits the dos Santos design and parameter values suggests that our findings can generalize to a *class* of HIV models.

3 Understanding the Efficiency via Profiling

We have created an implementation with instrumentation of the model as well as created multiple synthetic benchmarks to investigate specific aspects (e.g. memory limitations) of the algorithm. The implementation is done in Rust, a modern native language. For the performance measurements we use the Criterion library (Version 0.3). Measurements were done on a Workstation with an Intel Xeon E5-1630 v4 CPU using Rust 1.41. The code, benchmarks, and scripts to produce the plots can be found at https://osf.io/g5yah/.

3.1 Storing and Iterating Over Model States

Each cell in the model may be in one of 7 states (Healthy, Acute with $\tau \in \{0,1,2,3\}$, Latent, Dead). We store these states in one byte. Although fewer bits may be used, the small gain in space would be at the expense of increased computations to pack/unpack states. To support compiler-based optimizations, we store states in a contiguous (one dimension) array, which supports fast read/write operations.

Vectorization is an optimization where a program uses CPU specific instructions that can handle multiple data chunks in one step. This is a type of instruction level parallization also known as Single Instruction Multiple Data (SIMD). For improvements in the vectorization we use the ndarray (version 0.13) library which provides a matrix abstraction that we use for the implementation of our algorithms. Readers more familiar with Python than with Rust may think of this abstraction as similar to numpy, albeit more closely integrated into the language. We use the ndarray zip operator to iterate over the grid. Neighborhood checks are done via iterations based on indexes[1].

Note that this baseline implementation is not a naive one, as that would lead to unrealistically low performances. Our baseline implements a CA with several techniques including making an efficient use of the memory and optimizing the iteration schemes.

[1]By default this would not necessarily be fast in Rust. All typical array access via indices are bound-checked at runtime. This was circumvented for this implementation by using unsafe code.

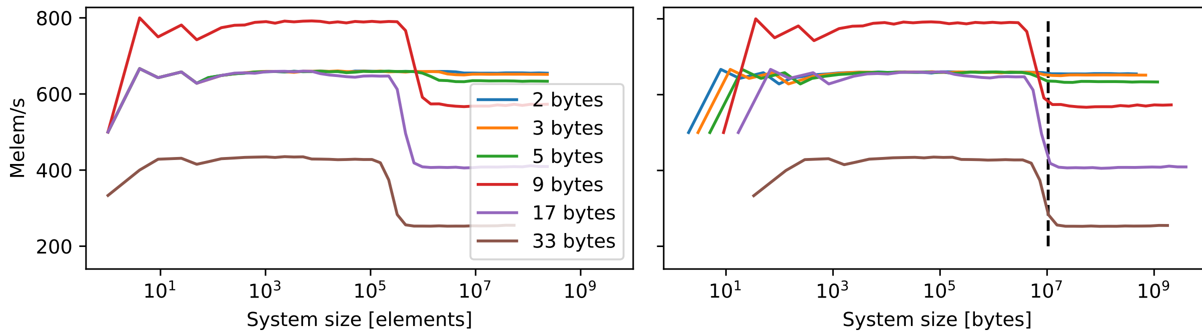

Figure 2: Throughput of different synthetic benchmarks for different number of cells. The different lines correspond to different data sizes. There is a different peak throughput due to different computational optimization, with 64bit integers shown in red as fastest due to the optimal native support and 256 bit integers requiring two separate operations. For some larger per cell memory needs we can see how CPU caches are no longer able to provide a performance boost when considering larger number of cells. For smaller per cell storage however even RAM memory rates are sufficient. The separation at 10MB (vertical dashed line) is consistent with the size of this CPUs L3 Cache. These results also indicate that our results are applicable to arbitrarily large systems as well.

3.2 Performance Metric

Our performance metric is the number of cells processed per second, given in million elements per second (Melem/s). We will be measuring this throughput at each step, as well as averaged across the whole 600 steps that are commonly conducted in an HIV CA simulation. This metric only accounts for the operation that are required in the simulation model, hence additional operations used for analyses (e.g., tracking the number of cells in each state) are not included in our performance metric.

3.3 Synthetic Benchmark and Maximum Theoretical Throughput

In this section we successively build up some synthetic benchmarks, to get an estimate for potential performance limitations. Firstly, to identify the maximum throughput that our machine can achieve, we create a synthetic benchmark consisting of an array randomly filled with states $S_1, S_2, \ldots$, which we replace using simple rules such as $S_1 \rightarrow S_5$ and $S_5 \rightarrow S_2$. This is the most minimal and simple version of this type of simulation model and implementation. The throughput we get for this benchmark is the limit no traditional implementation of an HIV model could surpass, since such models always involve more computations in their transitions (e.g., accessing neighbors, applying probabilities) Note that there are more advanced works that do not need to access the entire system, like Memoization (Gosper 1984), but those are beyond the scope of this paper.

As this benchmark involves very few 'hard' operations in contrast to HIV models, its throughput may be too high. We thus create a second benchmark to offer a tighter bound, by incorporating harder operations. Specifically, we make optimal code generation a little harder for the compiler by introducing a state S_{null}. This state is not created initially and none of the other states transitions to it, but its transition rule is very complex requiring change of a global state, similar to a random number generator or a counting mechanism. This limits the kind of optimizations a compiler and CPU can perform, like vectorization.

This already has a tremendous influence on the performance. For the first benchmark we get a rate of 788 Melem/s whereas the second yields only 156 Melem/s. We should stress again that the only difference between these two benchmarks is a state transition rule, which is never triggered but whose conditions make it harder to optimize the code..

For the purpose of comparison we have created a third synthetic benchmark with all identical cells, applying the same hard to vectorize kernel that yielded 156 Melem/s earlier we now get more than 1470 Melem/s. This is because the CPU can optimize at runtime for identical data. The increased mixing of the system therefore leads to slower performance.

A few of the transitions in the dos Santos model depend on probabilities. To understand the impact of these stochastic transitions on performances, we consider an extreme case in which half of all transitions are stochastic in a fourth and fifth benchmark. We modify the first synthetic benchmark model accordingly. Naturally this raises the question of how the random numbers are generated. We consider two cases, one with Rusts default, very high quality cryptographically secure ChaCha block cipher random number generator and one using the much faster Xoroshift random number generator. For these model implementations we get rates of 83 Melem/s and 134 Melem/s respectively.

The difference between the easy and hard to vectorize throughput (first and second) is already an indication that memory access is likely not the limiting factor for the performance of this computational problem (i.e. it is compute- rather than memory-bound). To confirm this we run a series of tests where we modify our first synthetic benchmark as such: In between the states of the dummy model we interweave counters that are incremented on every access. We use different data types to test for the memory limit. Results (Figure 2) show the effect of caches for the smaller grid sizes, but for a larger amount of cells and larger counters we run into the memory limit. This limit however is very far from the domain applicable here, as our memory load is very small. For any real model implementation, the transition rules are more complex and therefore putting even less strain on memory throughout. Therefore in the following we can focus on optimization for computational throughput.

3.4 Results for our baseline implementation

Our baseline implementation has a throughput of about 76 Melem/s, which is about 10% of the maximum as established by the first synthetic benchmark (Figure 2), and already about half of the more reasonable harder to vectorize second benchmark.

The throughput of the baseline implementation changes throughout the model run. After the first few steps (which show extreme model behavior), performance increases, then after about 200 steps decreases. We can explain this by looking at the simulation behavior (Figure 3). Initially, we have many healthy cells (Figure 3(a)–top), each of which requires an expensive neighborhood check. Later, there are fewer healthy cells, hence performances improve. However, after about 220 steps, performances degrade again even though the number of healthy cells (as well as all other cells) remains constant. This behavior can be understood by examining the heterogeneity in the simulation (Figure 3(b)). Initially the system is quite well ordered and homogeneous, as measured by the percentage of cells whose left neighbor has the same state. These are particularly easy to execute efficiently by the CPU.

Using faster (but lower quality) Xorshift random numbers leads to a throughput of about 91 Melem/s in comparison to 76 Melem/s before.

4 Algorithmic Performance Improvements

Results in the previous section have established that randomness and branching intensive code (e.g. neighborhood checks) are the limiting factors for performances in our implementation. In this section, we present three optimizations to improve performances: avoiding the over-generation of random numbers, reducing the number of conditional checks when looking at the states of neighboring cells, and a combination.

4.1 Random Number Calculation via Geometrical Distribution

In the traditional implementation of this CA model, the random numbers are sampled for every cell. For example, *every* dead cell requires the computation of a probability p_H to decide whether it is replaced with a healthy one. Our analysis of the rates used specifically in this model (Figure 1) reveal that stochastic events

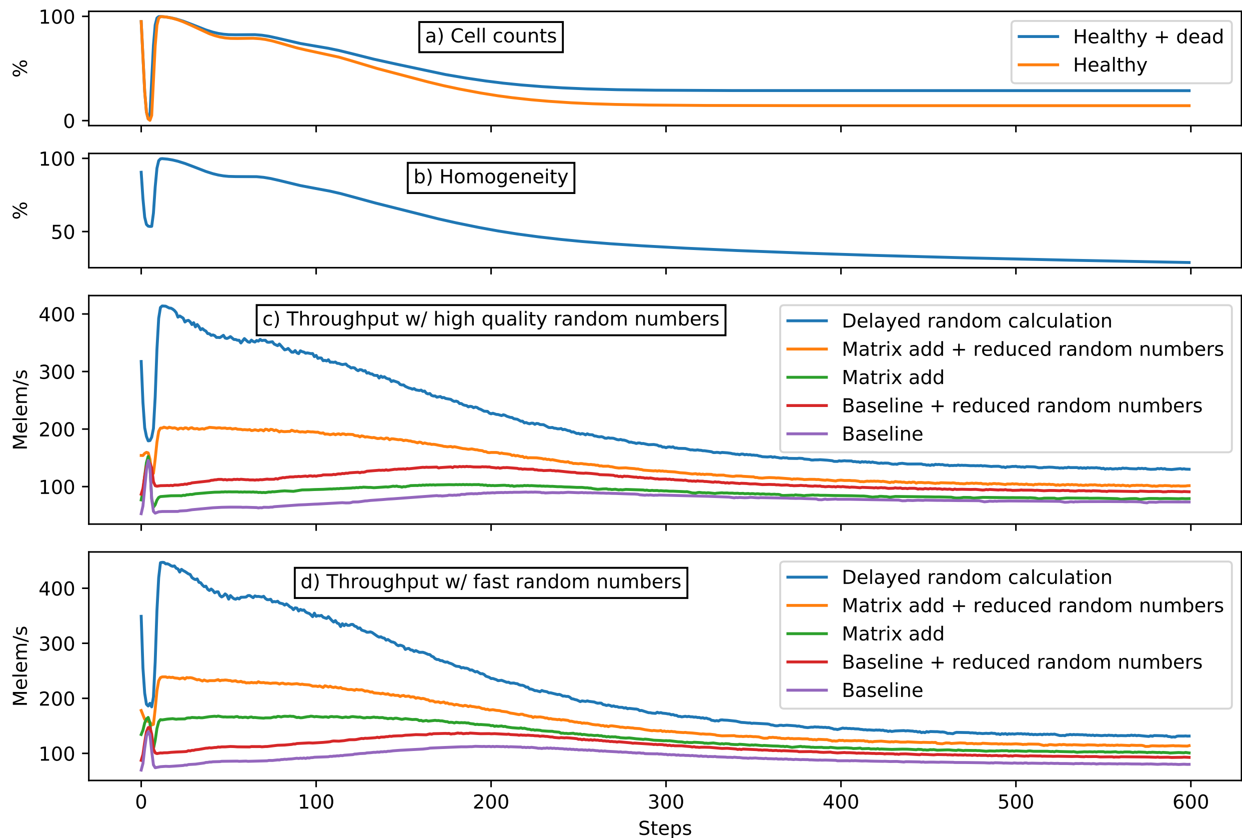

Figure 3: The simulator output is shown by counting healthy and dead cells (a) and the number of cells that have the same state as the cell to their left (b). The throughput is shown in two configurations: very high quality random numbers (c), or low quality but faster to compute (d). All runs use 10 replications and a grid of 1500×1500 cells. Step-averaged data including errors can be found in Table 1.

are either very rare (probability is close to zero) or very likely (probability almost one). Calculating the random probability for every cell is very costly. To lower this cost, we note that events are all independent, but follow the same Bernoulli process. For example if we have an event with probability $= 0.997$, a negative sample will occur on average only about every 333 samples. The number of negative samples between two positive samples follows the Geometric distribution. We only need to calculate one sample of that distribution (using a standard library) and then count until we have reached this number of occurrences. This works well for both very high as well as very low probabilities, as for those the number of avoided random number calculations is the highest.

4.2 Neighborhood checks via Matrix addition

Encoding the neighborhood checks using addition instead of iteration and equality checks has already been presented in the literature (Giabbanelli et al. 2020). However, these checks were still performed on demand. That is, a cell was first checked to know whether it was healthy, and then the addition was performed.

In the early phase of the simulation, most cells are healthy. To calculate the newly infected cells, all cells in the neighborhood of each cell are being tested. To optimize this calculation, we introduce vectorized additions. For that we take the entire grid as a matrix and add the eight shifted variants (for the Moore neighborhood) and add them to this matrix. These additions are very cheap, as we can use highly optimized (and vectorized) linear algebra libraries.

Random Numbers	Simulator	prob. transitions	throughput [Melem/s]	setup in %
high quality	baseline	every time	76.4 ± 0.1	
		geometric distr.	108.0 ± 0.2	
	matrix add	every time	88.9 ± 0.5	9.8 ± 0.5
		geometric distr.	131.8 ± 1.1	14.5 ± 0.7
	delayed random	n/a	182 ± 3	27.6 ± 0.7
high speed	baseline	every time	91.1 ± 0.1	
		geometric distr.	109.5 ± 0.2	
	matrix add	every time	126 ± 1	13.7 ± 0.7
		geometric distr.	148 ± 2	16.5 ± 0.7
	delayed random	n/a	186 ± 2	24.6 ± 0.8

Table 1: Average throughput of the different simulators and their configurations configurations. Setup time is the time doing matrix adding as well as the random transition for the delayed random variant. Taken from 10 replications at 1500x1500 cells with errors showing standard error of mean. The behavior of efficiency through the model run can be found in Figure 3.

Later on we can iterate over the entire grid without the need to use any complex access patterns, as we have already computed the neighborhood for every cell, which minimizes branching. As observed in the synthetic benchmarks, branching is detrimental to optimal throughput. In branched code, the CPU has struggles with branch prediction and vectorization can be less efficient. This optimization is expected to be the most helpful in the first third of the simulation where there still are many healthy cells.

4.3 Delayed random transitions

Our third optimization in a way combines the first two presented. To improve vectorization and execution speed we need to minimize branching. One type of branching we have is checking at the random transitions. Even if we do not need to compute a new random number (using the geometric distribution) we still might need to alter a counter and check if it is equal to zero.

In this third optimization we remove all stochastic transitions by rounding the probabilities to one or zero. Together with the matrix addition optimization outlined above, this implementation has almost no branching. The only check that remains is in the presence of a healthy cell, as we need to check whether the pre-computed neighborhood sum exceeds the threshold.

Simply removing stochastic transitions would significantly alter model behavior. Therefore we need to find a way to reintroduce these rare events where stochastic tests would have failed. Here we leverage that the number of times such a test passes or fails follows the binomial distribution. After a step has been executed, we then visit a number of cells sampled from the correct binomial distribution and reverse the step. This binomial distribution takes the probability of success per cell and the total number of cells as parameters. Therefore the optimization is still exact and not approximate.

4.4 Results for the Proposed Three Optimizations

We have implemented the optimizations presented above and the results can be found per step in Figure 3 and in step averaged form in Table 1.

Reducing the number of random numbers improves the baseline speed by 40% if high quality random number generators are used and by 20% if the fast random number generators are used. As expected this optimization mainly affects the beginning of the simulation where healthy and dead cells (i.e. those that need random numbers) make up a majority of the cells. When using the high quality random numbers, this optimization even outperforms the matrix add optimization.

Using the matrix addition for the environment checks also improves performance with throughput rates of up to 148 Melem/s. Again this also mainly affects the first third of the simulation where there are very many healthy cells that need to check their environment. In the later stages of the simulation this optimization only slightly improves upon the baseline. Performing this matrix addition takes only about 15% of the runtime, thus it is not a large contribution to runtime, even though potentially more work than is strictly needed is performed.

The delayed random calculation where random transitions are added afterwards is again a lot faster than the optimizations presented above. It reaches speeds in excess of 400 Melem/s leading to an average throughput of about 186 Melem/s. Performance mainly goes down with increased mixing in the system. Performing the delayed random calculation as well as using matrix addition for the testing the neighborhood takes about 25% of the total runtime, so the main iteration loop still dominates.

Overall, we see a 2.4 speed up, when comparing the fastest average throughput of 182 Melem/s with the baseline of 76.4 Melem/s for high quality random numbers.

4.5 Alternative Approaches

Modeling studies rarely report *negative* results, and HIV CA studies are no exception. However, it is important to know which other approaches could be attempted and the results that they lead to. Some of the results are negative *in our context* (e.g., no throughout improvements) but would be of interest in other cases (e.g., better bandwidth utilization). By disclosing these alternatives, we thus limit the risk that future teams will devote energy to approaches that were not fruitful. Other teams may be interested in some of the approaches below as they do lead to improvements other than throughput, which is the main consideration for this paper.

As the model has mostly a single path of transition (*Healthy* then various levels of $Infected$ then *Dead*), we implemented a **buffer free** variant of the simulation. However this has only reduced memory and bandwidth usage and neither of them are currently limiting performances. In practice, this sometimes even reduced performances due to a more complex environment check.

We have examined whether vectorization could be applied locally at the neighborhood level, i.e. **vectorizing the neighborhood check** every time we encounter a healthy cell. However, due to the small neighborhood tested (a Moore of 8 neighbors) as well as the irregularity of the pattern alignment in memory, this decreased performances. Note that there have been attempts to expand the neighborhood of HIV CA models to account for both proximal (cell-to-cell) and more distal (cell-free) infections (Giabbanelli et al. 2019). In such cases, the larger neighborhoods may be better candidates for explicit vectorization.

The dos Santos model seeks to replicate the first two phases of HIV in the absence of any treatment. Individuals thus necessarily have a larger proportion of infected cells later in the simulation (which would not necessarily be the case if therapy was modeled). Given that almost all neighborhoods contain an acutely infected cell in later parts of the dos Santos model run, we assessed the potential of optimizing the neighborhood look-up. For that, we created a **lookup data structure** that was a quarter of the size of the system, with each entry containing the sum of four cells. By checking in the lookup structure first, most neighborhood iterations could be avoided, however accessing the structure itself had a similar overhead, resulting in no speedup. This does not check the neighborhood explicitly, but if any of the four cells contain the an infectious cell, then checking becomes unnecessary.

5 Soundness of the CA Approach

The performance of simulation algorithms is important as an efficient simulation algorithm is necessary to conduct extensive experiments with a simulation model. Designed Experiments help us understand how a model responds or identify which (combinations of) parameter values are most important. By shedding light on the behavior of the model, such experiments contribute to building trust in the model's ability to provide useful answers for its intended application context.

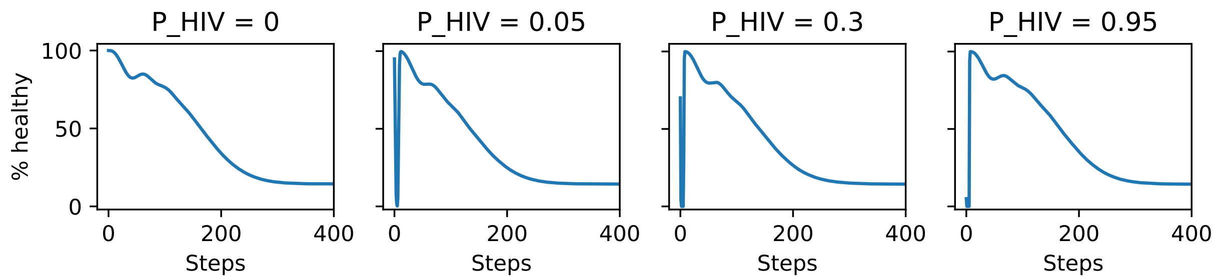

Figure 4: A change in the parameter P_{HIV} only affects the first few steps of the model. This is an artefact of the methodology.

Another essential expectation to build trust is that the simulation algorithm correctly implements the approach. However, another aspect of soundness is whether the approach is appropriate for the system and application context. The former requires comparing the simulation results of different simulation algorithms, which forms an intrinsic part of any performance study, independently of whether exact or approximate, spatial or non-spatial simulation algorithms are evaluated(Jeschke et al. 2011).

It is well known that synchronously updated cellular automata are problematic for many real world simulation applications (Schönfisch and de Roos 1998). Therefore, cellular automata models that simulate tumor growth use an asynchronous updating scheme instead: “These methods [cellular automata] usually update the lattice sites in a random order to reduce grid artifacts” (Metzcar et al. 2019). This is in contrast with the dos Santos model, where (emphasis added) “in one time step the entire lattice is *updated in a synchronized* parallel way” (dos Santos and Coutinho 2001). If a cellular automaton with a regular grid is executed synchronously (as is the case with the dos Santos line), some directions are favored over others. This leads to the known problem of anisotropic behavior (Schönfisch 1995): a direction-dependent speed of information spread throughout the system, usually favoring diagonals. In addition, fixed time steps in simulations as typical for cellular automata execution can introduce further errors. Therefore, fixed time steps should usually only be used if the system itself is one of regular steps. A discussion can be found in the literature, for example in Appendix 1A of (Law 2007).

5.1 Manifestation in this particular model

We now examine the particularities of this specific model (and its close relatives) to investigate whether some of these problems also apply here. The behavior of our model consists of 3 *Phases*. Initially the entire system goes through the complete cycle of infected, dead, and then again healthy, since about one in twenty cells is infected. Here we already have a particular artefact of the synchronous update, that is, that this spread basically dies out immediately. If we were to alter this initial infection proportion from 5% to 95% or 0% the model behavior after the first few step would not change at all. The rest of the trajectories remain the same. This can be seen in Figure 4.

Then there is a second phase, of regular patterns in the system. Here some cells randomly turn sick. Taking the infected cell as origin, waves of infection propagate through the system. However due to the mostly deterministic nature of the transition, the cells all get healthy again after one iteration of the cycle as shown in Figure 1. This is because the waves are very stable and therefore the dead cells form a barrier shielding the healthy cells from becoming infected again. These transitions are rarely avoided due to stochastic sampling (a fact which we leveraged for performance improvements).

On the very rare occasion, a transition does not take place multiple steps in a row, i.e. a cell remains healthy while its surroundings are infected. In this situation, we transition into the third phase, the random noise phase. This cell will be infected at the same time as its surrounding cells become healthy. Those cells in turn get infected, turn this particular cell healthy and so on, therefore the infection cannot die out

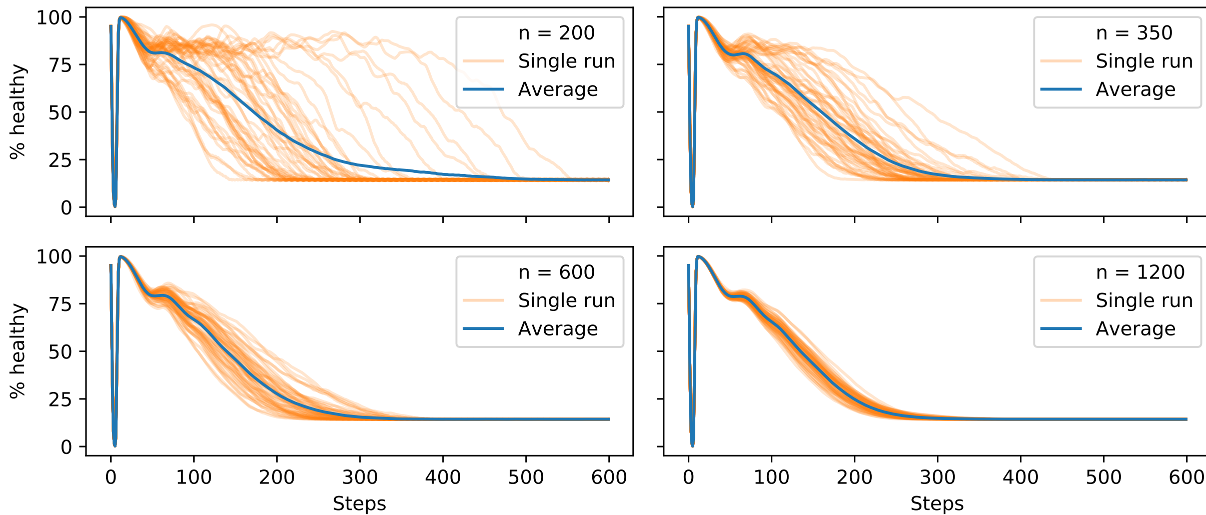

Figure 5: Larger system sizes do not change behavior, but lead to a convergence of individual runs due to spatial averaging. We compare a grid of $n \times n$ cells varying n between 200 and 1200. The percentage of healthy cells is plotted both for the individual runs as well as their average. For larger systems we only see more of a regression to the mean, but no additional insight into the model behavior. At least for the basic dos Santos model, larger models do not provide additional insight.

at this point. This is again an artefact of the synchronous update with high probabilities. This artefact will then serve as a contiguous source of infection waves in the system, forming irregular shapes and leading to a more and more random phase, which is the steady state. This artefact has already been described by the original authors.

Another question is how sensitive the behavior of the model is to changes in its size. The system is simulated for 600 steps. For larger systems, such as 1000x1000, a transition at one end of the system cannot affect a cell more than 600 cells away, since it has a maximum reach of 1 grid cell per time-step. Here, increasing the system size serves to average multiple potential trajectories. A system of 600x600 cells with periodic boundary condition will have the exact same average behavior as a system with 60000x60000 cells.Our performance results however, will be applicable to larger models as well, since our implementation is compute bound. This is the reason we supply throughput and not total runtime numbers.

5.2 Discrete event-based simulation as a solution to problems of soundness

Discrete event-based simulation appears as a natural remedy to counterbalance the problems introduced due to synchronous, step-wise computation of the cellular automata. Approaches such as Cell-DEVS and similar (Zeigler 1982; Wainer 2014) exploit the potential of discrete event simulation for cellular automata type models. They offer the possibility to explicitly associate states with sojourn times, and thus have a natural notion of time and can easily be adapted to arbitrary networks beyond grids. We have tried to implement the dos Santos Model as a discrete event simulation using exponential distributed sojourn times as well as explicit delays. Although the conceptual translation appears straightforward, the resulting model behavior differs significantly. Instead of the initial infection immediately dying out, we transition directly to the stable random state phase, where we have random mixing of all model states. Regular structures that shield infected cells from becoming sick do not occur when allowing transitions in continuous time.

Another indication that the observed behavior is the result of the synchronous, step-wise update is the question what happens if we reduce the step-size. One would expect if the time step is reduced from one week to one day and the probabilities per step accordingly adapted to receive the same results. However,

the smaller probabilities per time step allow for the wave fronts to be interrupted. As a consequence they dissolve quickly. Thus the second phase of the observed behavior will not occur.

6 Conclusion

We assessed the execution characteristics of the established dos Santos cellular automaton model from a performance standpoint. We implemented and tested several algorithmic variants that surpass the performance of an already optimized implementation for these type of CA models by a factor of up to 2.4. A crucial ingredient of the achieved speedup has been separating the model into a small stochastic part and a large deterministic part. The expensive deterministic part maps well onto vector execution of the CPU for this compute bound problem.

Analyzing the behavior of the model revealed several limitations. Retrospectively we observe, that the very characteristics we exploited to increase performance (synchronicity of execution with very rare stochastic variations within the model) are the same ones that also limits the soundness of the approach. Consequently, our novel performance study also emphasizes the importance of analyzing model characteristics and soundness not only of the algorithm but also of the approach.

REFERENCES

Caux, J., P. Siregar, and D. Hill. 2010. "Hash-life algorithm on 3D excitable medium application to integrative biology". In *Proceedings of the 2010 Summer Computer Simulation Conference*, 387–393. Society for Computer Simulation International.

Deutsch, A., S. Dormann et al. 2005. *Cellular automaton modeling of biological pattern formation*. Springer.

dos Santos, R. M. Z., and S. Coutinho. 2001. "Dynamics of HIV infection: A cellular automata approach". *Physical review letters* 87(16):168102.

Fisher, A., B. Adhikari, C. Chai, J. E. Morgan, V. K. Mago, and P. J. Giabbanelli. 2020. "Predicting the resource needs and outcomes of computationally intensive biological simulations". In *Proceedings of the 2020 Spring Simulation Conference*.

Giabbanelli, P. J., C. Freeman, J. A. Devita, T. Koster, and J. A. Kohrt. 2020. "Optimizing Discrete Simulations of the Spread of HIV-1 to Handle Billions of Cells on aWorkstation". In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*.

Giabbanelli, P. J., C. Freeman, J. A. Devita, N. Rosso, and Z. L. Brumme. 2019. "Mechanisms for Cell-to-cell and Cell-free Spread of HIV-1 in Cellular Automata Models". In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 103–114.

Golpayegani, G. N., A. H. Jafari, and N. J. Dabanloo. 2017. "Providing a therapeutic scheduling for HIV infected individuals with genetic algorithms using a cellular automata model of HIV infection in the peripheral blood stream". *Journal of Biomedical Science and Engineering* 10(3):77–106.

González, R. E., S. Coutinho, R. M. Z. dos Santos, and P. H. de Figueirêdo. 2013. "Dynamics of the HIV infection under antiretroviral therapy: A cellular automata approach". *Physica A: Statistical Mechanics and its Applications* 392(19):4701–4716.

Gosper, R. 1984. "Exploiting regularities in large cellular spaces". *Physica D: Nonlinear Phenomena* 10(1):75 – 80.

Granich, R. M., C. F. Gilks, C. Dye, K. M. De Cock, and B. G. Williams. 2009. "Universal voluntary HIV testing with immediate antiretroviral therapy as a strategy for elimination of HIV transmission: a mathematical model". *The Lancet* 373(9657):48–57.

Hillmann, A., M. Crane, and H. J. Ruskin. 2017. "A Computational Lymph Tissue Model for Long Term HIV Infection Progression and Immune Fitness.". In *AICS*, 245–257.

Jeschke, M., R. Ewald, and A. M. Uhrmacher. 2011. "Exploring the performance of spatial stochastic simulation algorithms". *Journal of Computational Physics* 230(7):2562–2574.

Kaplan, E. H. 1989. "Needles that kill: modeling human immunodeficiency virus transmission via shared drug injection equipment in shooting galleries". *Reviews of infectious diseases* 11(2):289–298.

Kougias, C. F., and J. Schulte. 1990. "Simulating the immune response to the HIV-1 virus with cellular automata". *Journal of Statistical Physics* 60(1-2):263–273.

Law, A. M. 2007. *Simulation Modeling & Analysis*. 4 ed. New York, NY, USA: McGraw-Hill.

Metzcar, J., Y. Wang, R. Heiland, and P. Macklin. 2019, November. "A Review of Cell-Based Computational Modeling in Cancer Biology". *JCO Clinical Cancer Informatics* (3):1–13.

Moonchai, S., and Y. Lenbury. 2016. "Investigating Combined Drug and Plasma Apheresis Therapy of HIV Infection by Double Compartment Cellular Automata Simulation". *International Journal of Computer Theory and Engineering* 8(3):190.

Pandey, R. B., and D. Stauffer. 1990. “Metastability with probabilistic cellular automata in an HIV infection”. *Journal of statistical physics* 61(1-2):235–240.

Poleszczuk, J., and H. Enderling. 2014. “A High-Performance Cellular Automaton Model of Tumor Growth with Dynamically Growing Domains”. *Applied Mathematics* 05(01):144–152.

Precharattana, M. 2016. “Stochastic modeling for dynamics of HIV-1 infection using cellular automata: A review”. *Journal of bioinformatics and computational biology* 14(01):1630001.

Precharattana, M., W. Triampo, C. Modchang, D. Triampo, and Y. Lenbury. 2010. “Investigation of spatial pattern formation involving CD4+ T cells in HIV/AIDS dynamics by a Stochastic cellular automata model”. *International Journal of Mathematics and Computers in Simulation* 4(4).

Rana, E., P. J. Giabbanelli, N. H. Balabhadrapathruni, X. Li, and V. K. Mago. 2015. “Exploring the relationship between adherence to treatment and viral load through a new discrete simulation model of HIV infectivity”. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 145–156.

Rybacki, S., J. Himmelspach, and A. M. Uhrmacher. 2009. “Experiments with single core, multi-core, and GPU based computation of cellular automata”. In *2009 first international conference on advances in system simulation*, 62–67. IEEE.

Salguero, A. G., M. I. Capel, and A. J. Tomeu. 2019. “Parallel Cellular Automaton Tumor Growth Model”. In *Practical Applications of Computational Biology and Bioinformatics, 12th International Conference*, edited by F. Fdez-Riverola, M. S. Mohamad, M. Rocha, J. F. De Paz, and P. González, 175–182. Cham: Springer International Publishing.

Schönfisch, B. 1995, February. “Propagation of fronts in cellular automata”. *Physica D: Nonlinear Phenomena* 80(4):433–450.

Schönfisch, B., and A. de Roos. 1998. “Synchronous and Asynchronous Updating in Cellular Automata”. In *Cellular Automata: Research Towards Industry*, edited by S. Bandini, R. Serra, and F. S. Liverani, 42–46. London: Springer London.

Shi, V., A. Tridane, and Y. Kuang. 2008. “A viral load-based cellular automata approach to modeling HIV dynamics and drug treatment”. *Journal of theoretical biology* 253(1):24–35.

Wainer, G. A. 2014. “Cellular Modeling with Cell-DEVS: A Discrete-Event Cellular Automata Formalism”. In *Cellular Automata*, 6–15. Cham: Springer International Publishing.

World Health Organization 2020. “Global Health Observatory (GHO) data”.

Zeigler, B. P. 1982, June. “Discrete event models for cell space simulation”. *International Journal of Theoretical Physics* 21(6-7):573–588.

AUTHOR BIOGRAPHIES

TILL KÖSTER is an research associate and PhD student in the modeling and simulation group at the University of Rostock. His research with the DFG ESCeMMO project focuses on the efficient simulation of cell-biological multi-level models. He holds Masters degrees in both Computer Science and Physics. His email is till.koester@uni-rostock.de.

PHILIPPE J. GIABBANELLI, Ph.D., is an Associate Professor in the Department of Computer Science & Software Engineering at Miami University (USA). His research interests include network science, machine learning, and simulation. He has directed projects in developing, verifying, and optimizing cellular automata for HIV. He currently serves as track chair for the 2020 Spring Simulation Conference, and program chair for the 2020 ACM SIGSIM Principles of Advanced Discrete Simulations (PADS) conference. His email address is giabbapj@miamioh.edu. His website is https://www.dachb.com.

ADELINDE M. UHRMACHER is head of the modeling and simulation group at the University of Rostock. Her research focuses on the development of modeling and simulation methods for multi-level systems and their application in areas such as cell biology, demography, or ecology. Methodological developments of her group include domain-specific languages for modeling and simulation, methods for automatically generating and executing simulation experiments, efficient simulation algorithms, and methods that assist in conducting and documenting simulation studies. She was editor in chief of SCS Simulation (2000-2006) and ACM Transactions of Modeling and Computer Simulation (2013-2019). Her e-mail is adelinde.uhramcher@uni-rostock.de