Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, eds.

COMPARING THE EFFECT OF CODE OPTIMIZATIONS ON SIMULATION RUNTIME ACROSS SYNCHRONOUS CELLULAR AUTOMATA MODELS OF HIV

Junjiang Li
Philippe J. Giabbanelli

Department of Computer Science &
Software Engineering
Miami University
501 E High St
Oxford, OH 45056, USA

Till Köster

Institute for Visual and Analytic Computing
University of Rostock
270 Konrad Zuse Haus
18059 Rostock, GERMANY

ABSTRACT

Models developed by domain experts occasionally struggle to achieve a sufficient execution speed. Improving performances requires expertise in parallel and distributed simulations, hardware, or time to profile performances to identify bottlenecks. However, end-users in biological simulations of the Human Immunodeficiency Virus (HIV) have repeatedly demonstrated that these resources are either not available or not sought, resulting in models that are developed through user-friendly languages and platforms, then used on workstations. This situation becomes problematic when performances cannot cope with the salient characteristics of the phenomenon that is modeled, as is the case with cellular automata (CA) models of HIV. In this paper, we optimize the Python code of CA models of HIV to scale the number of cells handled by a simulation on a workstation commonly available to end-users. We demonstrate this scalability via five HIV CA models and compare these results to assess how modeling choices can impact runtime.

1 INTRODUCTION

In fields such as cellular biology, individual-based models are often developed through user-friendly platforms (e.g., MATLAB, RepastS, or NetLogo) or languages (Willem et al. 2017). As preferred platforms and languages change over time, we now observe an increased reliance on Python, not only to assist with analytical efforts such as visualizations (Ortigoza et al. 2020) but also to run the simulations (Ghosh and Bhattacharya 2020). It is indeed relatively simple to produce a working code for a Cellular Automaton (CA) in Python, as this activity is often given to students either in an introductory course (Giabbanelli 2012; Giabbanelli and Jackson 2021) or an advanced beginners' level course (Staubitz et al. 2016; Giabbanelli and Mago 2016). This emphasis on simplicity of implementation has several benefits: subject-matter experts can directly produce models, while in more complex projects modelers can focus on design and validation rather than implementation (Vendome et al. 2020). The trade-off is often clear: "the simplicity of usage was given higher priority than performance" (Guan and Clarke 2010). Intuitively, this is an acceptable trade-off when a model can be small while remaining adequate for the problem at hand. In the case of CA, 'small' covers several dimensions: low number of cells (i.e. the height and weight of the grid), low number of update steps, and low number of repeats (either because there is no randomness or because the desired confidence interval requires few repeats). However, there are several problems that would be massive in all three dimensions and for which the common reliance on user-friendly platforms leads to severe limitations for models. Our paper aims to bridge this gap by demonstrating that Python-based solutions can achieve high performances for cellular automata on a workstation, thus enabling users to better cope with the needs of their problem while continuing to work in their preferred environment.

In this paper, our application is the epidemiological problem of modeling the Human Immunodeficiency Virus (HIV) within a human host. There are two reasons for which this application exemplifies the matter of performances for CA code in Python. First, the problem is massive in all three aforementioned dimensions. Consider that each cell of a HIV CA corresponds to a human white blood cell fighting infections, known as a $CD4^+$ T cell. There is an estimated total of 10^{11} $CD4^+$ T cells in a whole body (Zhang et al. 1998). We would also need a *massive number of update steps* to closely follow the biological course of HIV since this disease requires treatment for life (i.e. long total time span) and many important events are in the scale of hours to days (Murray et al. 2011). The large *number of repeats* observed in several models (Giabbanelli et al. 2019) echoes the "large between-subject variation in estimated viral dynamic parameters [...] even after accounting for variations in drug exposure and drug susceptibility" (Wu et al. 2005).

Second, several issues are caused by the disconnect between the small dimensions effectively managed by the implementations of HIV CA models and the massive dimensions that characterize the biology of HIV. Despite a development spanning almost 20 years, previous models have used grids of equal height and weight in the order of hundreds of cells: 100 x 100 (Precharattana et al. 2010; Rana et al. 2015), 250 x 250 (Golpayegani et al. 2017), 500 x 500 (Hillmann et al. 2017; Hillmann et al. 2020) or 700 x 700 (dos Santos and Coutinho 2001; González et al. 2013). The exception is a 38 x 38 grid (Jafelice et al. 2015). Modelers are aware that this captures at most half a million cells instead of hundreds of billion (Hillmann et al. 2017). An issue is then the inability to set enough cells aside as latent reservoirs, which is a defining feature of the disease and an important object of current research in immunology (Sengupta and Siliciano 2018). From a modeling standpoint, having too few cells means that the virus has to be artificially 'slowed down' to avoid simulating implausible outcomes by using the same propagation rates as in reality but over a much smaller space. The inability to have enough cells thus results in the inability to follow the necessary temporal resolution for biological events (i.e. a spatial mismatch produces a temporal mismatch) (Giabbanelli et al. 2019). Finally, since models already struggle to cope with spatial and temporal dimensions, it is no surprise that they do not account for even more computationally intensive aspects such as viral mutations. This drastically reduces their clinical relevance, since viral mutations and the drug resistance conferred are "one of the major dangers that should be carefully investigated" (Ngo-Giang-Huong and Aghokeng 2019).

This paper builds on a recent article (Giabbanelli et al. 2020), which showed the first Python implementation that could run over six billion cells within ten minutes on a workstation. This article was devoted to optimizing *one* model. In contrast, the contributions of the present article include *(i)* the development of optimization schemes that work across a variety of models, hence creating reusable optimizations for developers of HIV CA models; and *(ii)* an assessment of how changes in modeling choices can impact performances based on *five* models.

The remainder of the paper is organized as follows. In section 2, we provide a brief background on the CA models and techniques occasionally used to optimize CA; we refer the reader to (Giabbanelli et al. 2019) for detailed discussions about the design of each model. Our methods are detailed in section 3 and results are presented in section 4, then discussed in section 5.

2 BACKGROUND

2.1 The Five HIV Models

Based on a survey of the literature and a selection of unique features (Giabbanelli et al. 2019), we selected five HIV models to optimize (Moonchai and Lenbury 2011; Zorzenon dos Santos and Coutinho 2001; González et al. 2013; Rana et al. 2015; Precharattana et al. 2010). In line with previous research, we will refer to each model by its first author, so "the dos Santos model" refers to the model published in (Zorzenon dos Santos and Coutinho 2001). In the remainder of this section, we give an informal definition of the type of CA under consideration. Furthermore, we will briefly discuss the common features shared by all five models, and illustrate the operation of one model as an example.

In all of the five models, the CA is represented as a regular $n \quad n$ 2D lattice (i.e. a grid) of cells that hold discrete *states*, representing their status with respect to HIV (e.g., healthy with or without therapy, infected at various stages, dead). Each cell interacts with a small subset of all cells around it (called its *neighborhood*) and may transition to another state based on *transition rules*. All of the five models adopt the Moore neighborhood, which refers to the eight surrounding cells. The definition of neighborhoods for cells on the boundaries of the grid is called the *boundary condition*. Most models use the *periodic boundary condition* (Figure 1), such that an infection reaching an edge can now wrap around the CA to continue its course. The one exception is the Rana model, which uses the *fixed boundary condition*. Finally, in all five models, transition rules are applied in steps. The models are synchronous hence all cells are updated simultaneously at each step. In other words, the update of a cell *within* a single time step does not immediately affect other cells: the updated cell will only be visible and usable by other cells for the next update. The total number of steps is 600 for all models.

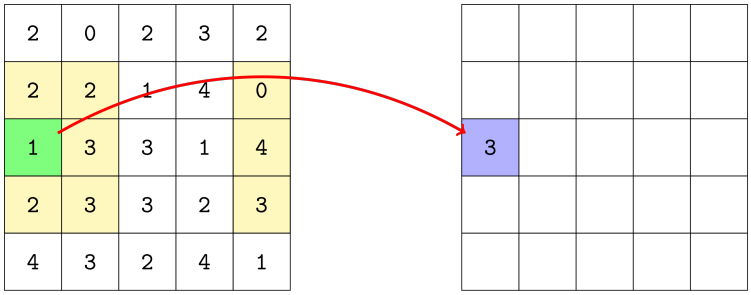

Figure 1: Illustration of CA definitions. The model has a grid size of $n \quad n$ with $n = 5$. The possible states of cells are numbers `0` to `4`. For the green cell, its Moore neighborhood under the periodic boundary condition is highlighted in yellow. When a transition rule is executed (red arrow), a cell with state `1` changes into `3`.

To illustrate typical rules, we will now focus on the specific example of the dos Santos model. Because the remaining four models build on it, illustrating the rules and execution of the dos Santos model serves to understand common design features in other models. Note that the dos Santos model and its successors do not take into account important biological features of HIV, such as mutation and drug resistance, in part due to the performance constraints addressed in this paper. As such, their focus is often on reproducing the qualitative trends in viral load rather than on making highly accurate clinical predictions. As mentioned before, the dos Santos model considers the Moore neighborhood and the periodic boundary condition. Furthermore, there are four cell states in the model, which are healthy (H), acute infected (I_1), latent infected (I_2), and dead (D). Four transition rules (one per state) describe the evolution of the model:

1. $H \not\to I_1$ if (i) there is at least one I_1 cell in its neighborhood, or (ii) there are at least c many I_2 cells in its neighborhood. Otherwise, $H \not\to H$. This rule simulates an infection from direct neighbors, which is known as a 'cell-to-cell' infection by contrast to 'cell-free' infection via the bloodstream.
2. $I_1 \not\to I_2$ if the cell has been in state I_1 for t_1 time steps. If not, $I_1 \not\to I_1$. This rule simulates the immune suppression of infected cells.
3. $I_2 \not\to D$ if the cell has been in state I_2 for t_2 time steps. This rule reflects the fact that infected cells will be killed by the immune system.
4. $D \not\to D$ with probability $(1 \quad P_{\text{repl}})$. Else, $D \not\to I_1$ with probability $P_{\text{repl}} \quad P_{\text{infec}}$, or $D \not\to H$ otherwise. This rule represents the replenishment of new cells, possibly as infected cells.

By default, the parameters are $c = 4$, $t_1 = 4$, $t_2 = 1$, $P_{\text{repl}} = 0.99$, and $P_{\text{infec}} = 1 \quad 10^{-5}$. In order for the simulation to start, *some* cells have to be infected just in the same manner as a forest fire simulation

would need burning trees to begin. We thus seed the simulation by iterating through all cells in an initially infection-free grid and manually set their states to I_1 with probability $P_{\text{HIV}} = 0.05$.

The aggregate behavior as well as the 95% Confidence Interval (CI) of the model is given in Figure 2, using our code to closely match the same figure by the original authors (Zorzenon dos Santos and Coutinho 2001). To further exemplify the execution of the model, snapshots of a single run at time steps 50, 200, and 400 are plotted in Figure 3 showing the entire grid (top row) or focusing on a subset to detail the shapes (bottom row).

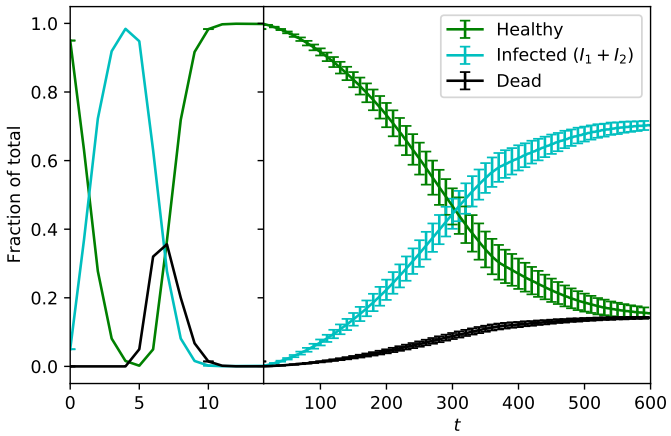

Figure 2: Percentage of types of cells as a function of time in the dos Santos model on a grid of 700 700. Error bars represent the 95% CI using 100 replications.

2.2 Optimization Techniques

Gosper proposed a powerful *memoization* algorithm (Gosper 1984), motivated by the famous deterministic CA of Conway's game of life. The memoization algorithm saved both memory and computation time compared to a basic implementation, while also improving the scaling behavior of the CA. This is achieved by exploiting the regularities of CAs at small scales and doing away with storing all cells in a contiguous array. Instead of computing and storing multiple copies of the same configuration, *only the first occurrence* is computed, and reference to the resulting grid is shared for any future repetitions (Gosper 1984).

A synchronous CA is a typical example of an "embarrassingly parallel" workload since the synchronous update allows to compute the next state of each cell independently based on a local neighborhood. Specifically, rules can be applied to different cells concurrently with no side effects by using a lattice to read the current state of each cell and another lattice to store their next state. This aspect was detailed in the seminal volume "Cellular Automata: A Parallel Model" (Delorme and Mazoyer 2013) and is used in simulations that leverage GPU/CPU architectures for large-scale parallelism (Guan et al. 2016; Gibson et al. 2015).

In terms of applications in HIV simulations, (Köster et al. 2020) profiled the dos Santos model to identify bottlenecks in CA simulations. The authors identified several factors closely tied to the runtime by using controlled synthetic benchmarks emulating the rules of the dos Santos model, as well as several test optimizations. Their results indicate that the generation speed of random numbers and the number of healthy cells (which have the most computationally intensive transition rules) produced by the model have the most weight, whereas memory access pattern has minimal influence. The largest performance gains could be made by exploiting the extreme (near deterministic) values for the probabilities in the system, thus reducing the amount of branching in the code and the number of random values that need to be computed.

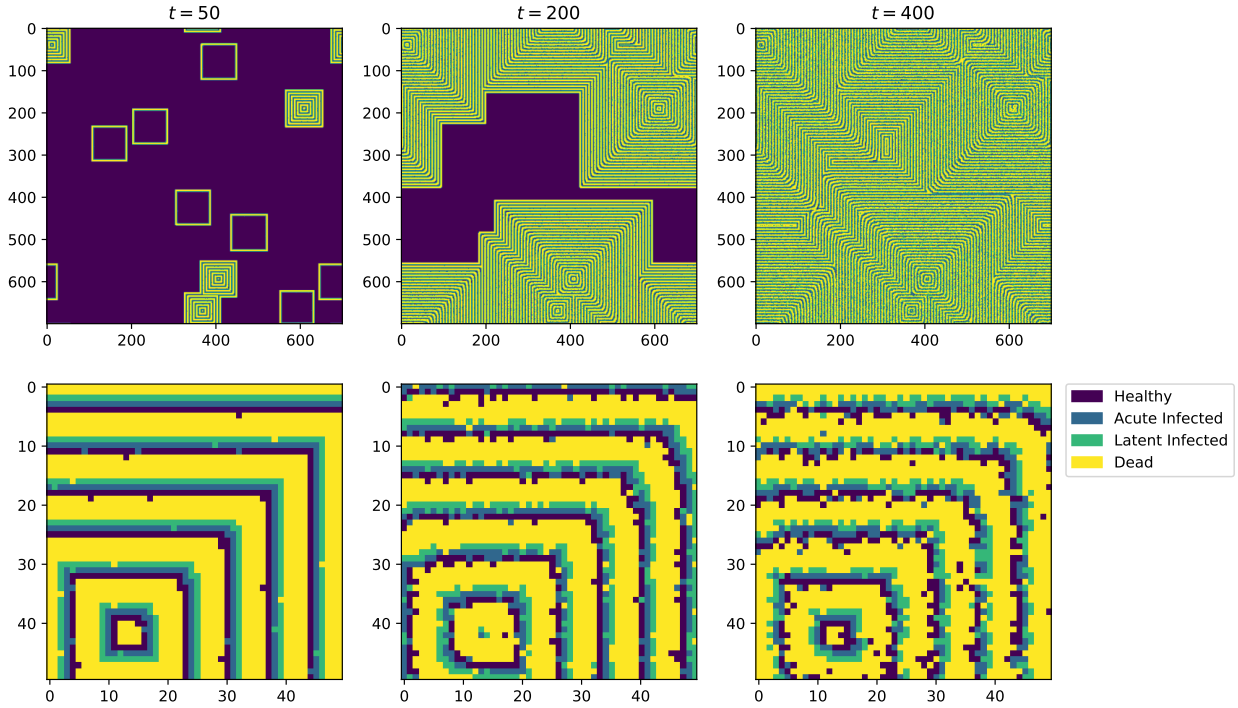

Figure 3: Snapshots of a single run on the dos Santos model on a grid of 700 700 at time steps 50, 200, and 400. The whole grid is shown at the top and its top-left corner is magnified in the figure underneath to observe how boundaries change during the simulation.

3 METHODS

3.1 Optimization Methods

We considered five optimizations, which are ① Just-In-Time (JIT) compilation, ② parallelization, ③ fast pseudo random number generators (PRNG), ④ simplified neighborhood checks, and ⑤ pointer swapping. Each optimization will be detailed in subsections 3.1.1 to 3.1.5. Each optimization can be enabled independently of the others, but for one exception: optimization ② is only meaningfully achievable when optimization ① is enabled due to the Global Interpreter Lock (GIL) in the Python language. Although we presented all five implementations previously (Giabbanelli et al. 2020), they were tailored to the dos Santos model. In contrast, we now abstract them such that they can be readily applied to *any* CA HIV model instead of being manually crafted for one particular model. The difference is particularly visible for optimization ④, which is now algorithmically constructed.

3.1.1 Just-In-Time (JIT) Compilation

Python is often compared to languages such as C++ on the basis of being an interpreted language and hence being slower than compiled languages. Compiling Python into statically typed, optimized machine code is thus essential to bridge this gap. We use the open source JIT compiler Numba for its ease of integration with existing Python code. To specify that a function should be compiled into machine code, one simply needs to mark the Python function with the decorator `@jit`. Upon the first call to a `@jit`-decorated function, Numba translates the Python function into C/C++ code with types inferred at call time, and compiles the generated code using the industry-standard LLVM compiler. According to the authors, "Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN" (http://numba.pydata.org/). The improvement is most expected in loops, which can be compiled a single

time instead of being repeatedly interpreted by the Python interpreter. This is particularly useful for cellular automata since the lattice containing cell states must be looped through and updated frequently.

3.1.2 Parallelization

On a regular workstation, possibilities for parallelism are more limited than on a compute cluster (leveraging multiple nodes) or in settings equipped with many CUDA cores (leveraging graphical processing units). The main opportunity explored in this paper is the use of multithreading such that different parts of a lattice can be updated by their own thread in order to significantly reduce the amount of time needed to compute a step. We rely on the auto-parallelization feature (similar to using an OpenMP directive) of Numba to split tasks into multiple threads, each of which is processed by a different core in the CPU.

Multithreading in Python faces a limitation: the GIL ensures that only one thread can manipulate objects or use the Python API at a given time. This makes the code execution safe since there can be no thread interferences, but it is also restrictive as it prevents multithreading and hence limits the execution speed of parallel workloads. Fortunately, Numba provides the `nopython` compilation option, which ensures the compiled function will not access the Python API during execution, effectively sidestepping the GIL.

3.1.3 Fast Pseudo Random Number Generators (PRNG)

Both the Python API and Numba use the Mersenne Twister PRNG to generate random numbers. Since dead and healthy cells transform based on probabilistic rules, increasing the generation speed may decrease runtime, particularly in later parts of a simulation run when more dead cells are present. The Xorshift family of PRNGs repeatedly applies a sequence of shift and xor operations, which are fast to compute. As these PRGNs fail several statistical tests, they have been improved through other operations, resulting in "scrambled Xorshift generators". Xoroshiro128+ is one of the latest ones (Blackman and Vigna 2018), serving as the successor to the xorshift128+ adopted in the Chrome browser. The shift/rotate-based linear transformations used by xoroshiro128+ allow this PRNG to generate values faster than most popular PRNGs, and the results are also of a higher statistical quality. The code originally developed in C (http://xoroshiro.di.unimi.it/xoroshiro128plus.c) was later ported to Python by Numba as part of the library for random number generations on CUDA GPU (https://github.com/numba/numba/blob/master/numba/cuda/random.py). We re-purposed the Numba code for our HIV CA models by removing the CUDA dependencies.

When using the default random number generator, the generator's state is managed in the thread-local memory, which cannot be easily accessed in Python. When using our own random number generator, we thus store a separate randomly initialized state for each chunk of parallel work, that is, each row of the CA. These chunks are assigned to different threads by Numba as part of its workload balance scheme.

3.1.4 Simplified Neighborhood Checks

In contrast with the three aforementioned optimizations, which can be applied to any synchronous and stochastic CA, this optimization needs to be adjusted based on the model. That is, this optimization is applicable if the CA model has exactly one neighborhood driven transition rule with the following form: cell i with state S transitions to state T if there is either at least 1 cell of state T or c ($2 \quad c \quad 8$) cells of state T' among the Moore neighbors of cell i. In the 5 models we consider here, this rule manifests as the transition from healthy cells to acutely infected cells under the presence of acute or latent infected cells, and $c = 4$. Traditionally, this check is implemented by visiting all neighbors and counting the number of cells with states T or T', in which *up to 16 comparisons* can be performed. We aim to reduce the number of checks by encoding states as numbers in such a way that when we sum up the encoded values of all neighbors of a cell, *only one comparison* against a chosen threshold a is needed to determine if the transition rule is satisfied. Our solution is explained in the next paragraph and then we provide an example.

We denote by $E : S \; ! \; Z$ the encoding function, where S is the set of states. Intuitively, E specifies how to replace all of the states (e.g., 'healthy', 'sick') by integer values (e.g., 5, 2). There are two cases:

the states T and T^0 which trigger the transition need to be replaced carefully, while all other states can be mapped to the first few integers in any order. Formally, let $U = S \; nfT, T^0g$ be the set of all states other than T and T^0. We number these states consecutively from 1 to jUj, that is, $E(U) = Z_{jUj} = f1,2,\ldots,jUjg$, where $E : U \; ! \; Z_{jUj}$ is bijective. The order of the numbering does not matter. The encoding of T and T^0 need to be more carefully designed as the occurrence of these two states dictate whether the transition rule is applied. Suppose for now that there are no cells of state T. To distinguish between having $c \quad 1$ many T^0 neighbors (in which the transition rule *will not* apply under our assumption) and having at least c many T^0 neighbors (where the transition rule *will* apply), we can assign to T^0 a number $E(T^0)$ large enough such that the sum of any configuration involving $c \quad 1$ or less T^0 neighbors will be less than $cE(T^0)$. Consequently, $E(T^0)$ satisfies the following inequality:

$$cE(T^0) > (c \quad 1)E(T^0) + (8 \quad (c \quad 1))jUj, \tag{1}$$

from which we can choose $E(T^0) = (9 \quad c)jUj + 1$. To finish the construction, we choose $E(T) = cE(T^0)$ and set the threshold a to the same value. As a result, the transition rule is applied to a cell if and only if the sum of encoding of all its neighbors is at least a.

For a concrete example, consider the dos Santos model in which there are four cell states H, D, I_1 and I_2 that stand for healthy, dead, acute infected and latent infected, respectively (c.f. section 2.1). Following the notation above, $U = fH,Dg$, $T = I_1$, $T^0 = I_2$, and $c = 4$. As there are two states fH,Dg other than infected, we assign them the numbers 1 and 2; that is, $E(D) = 1$, $E(H) = 2$. Then, $E(I_2) = (9 \quad 4) \quad 2+1 = 11$, and $a = E(I_1) = 4 \quad E(I_2) = 44$. Figure 4 exemplifies the use of simplified neighborhood checks.

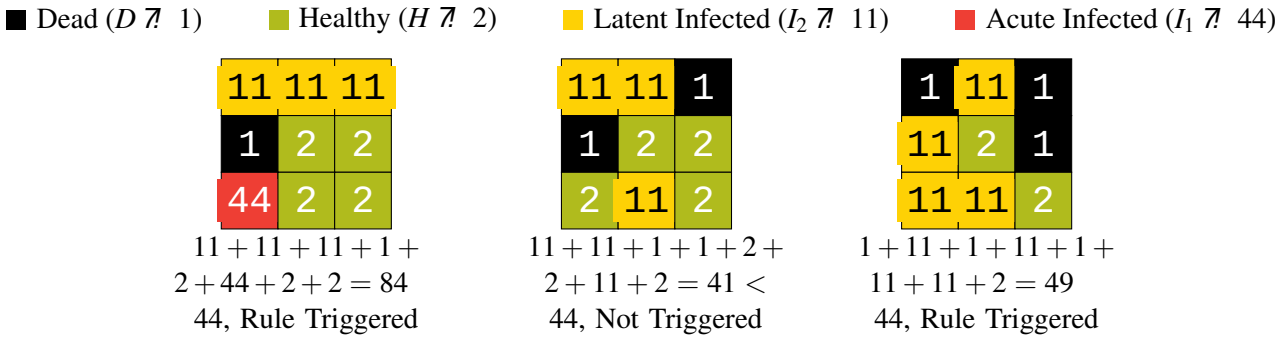

Figure 4: Examples showing the operations of simplified neighborhood checks for the center cell. Encodings are calculated using cell states listed in the dos Santos model.

3.1.5 Pointer Swapping

Programmatically, to ensure that all updates within one step are done independently, computation results are written to an intermediate buffer array (`buffer`) of the same shape as the CA grid (`grid`). When all cells have been updated, the values in `buffer` are normally copied over to `grid`. However, since the values in `buffer` are not important once they have been copied, we do not need to iterate through the `buffer` and copy each value onto `grid`. We circumvent this costly copy operation by swapping the references of `grid` and `buffer` using the pythonic construct "`grid, buffer = buffer, grid`".

3.2 Implementation and Experimental Set-up

We implemented all possible combinations of the optimizations discussed in the previous subsection, for all five HIV models. To examine the benefits and the generality of these optimizations, we computed the maximum grid length (i.e. the maximum n of an $n \quad n$ grid) that can be simulated within 10 minutes on a

workstation for each combination of optimizations and each of the five CA HIV models. This subsection details the implementation and approach to timing, while results are provided in the next section.

To examine the interactions between the optimizations, we seek to measure performances of all five models on all possible combination of optimizations. To avoid the error-prone approach of manually writing the scripts for all combinations, we create a more *systematic and reusable solution* whereby each script is automatically generated based on the optimizations to enable. For simplicity, each of the automatically generated scripts for a given combination of optimizations is denoted using 5-digit binary numbers of the form $O_1O_2O_3O_4O_5$, where $O_i = 1$ means that optimization ⓘ is enabled, and $O_i = 0$ means ⓘ is disabled.

As explained in 3.1.2, there is one *invalid* combination: ② is enabled but optimization ① is not. Since combinations $01xyz$ where x, y, z 2 f0, 1g are invalid, this leaves 2^5 $2^3 = 24$ possible combinations of optimizations to implement for each HIV model. To automatically generate the 24 possible scripts, we use the template language provided by the `jinja2` library. At generation time, appropriate Boolean flags are passed to the template processor indicating the optimizations to enable, then the optimized scripts are written instead of the unoptimized ones. The code listing (Figure 5) gives an excerpt of the template file. If we render this file with `optimizations['swap'] = True`, line 2 will appear in the rendered file.

```
1  {% if optimizations['swap'] %}
2          grid, buffer = buffer, grid
3  {% else %}
4          grid[:] = buffer[:]
5  {% endif %}
```

Figure 5: Excerpt of the template file to toggle optimizations

To quantify the improvements brought forth by the optimizations, we examine the throughput of the differently optimized scripts by finding the largest grid length n of an n n grid so that the entire simulation finishes within 10 minutes, with a tolerance for error of 1 minute. One approach would be to use binary search. Starting from a value b, we keep doubling b until the run time of the b b CA exceeds 10 minutes. Then, the desired grid length is contained in the interval $[0, b]$, and binary search can be employed. However, binary search is only effective when element access takes $O(1)$ time, whereas checking whether an n n CA finishes within 10 minutes (constituting an "element access") has time complexity $\Theta(n^2)$. As a result, using a binary search in the experimental set-up would be inefficient. Through experimentation, we noticed that the run time t is almost directly proportional to n^2 for grid sizes that are sufficiently large yet small enough to finish within 10 minutes. Therefore, if we approximate $t = an^2$, determining a will allow us to estimate the largest n runnable under 10 minutes. The procedure leveraging this observation to efficiently find n is described in Algorithm 1. Lines 4–6 ensure that n is sufficiently large so that the approximation $t = an^2$ is accurate. From there on, lines 7–10 iteratively refine the estimate of n using timing information of the previous run. For each combination of optimizations and of each model, Algorithm 1 is ran 10 times (i.e. 10 repeated measurements) to produce confidence intervals.

4 RESULTS

To provide full transparency and support the replication of results, our scripts are provided at https://osf.io/uxmkv/. Readers wishing to replicate this section can use the scripts located under WinterSim'21-Applications. All results are obtained with dual Intel Xeon Gold 6126 processors through a university computing cluster (Redhawk at Miami University), which individually resemble professional workstations locally available to users of HIV CA models. Each processor has 12 physical cores at 2.60 GHz, and all 24 cores are used in the benchmark. Results are summarized in Figure 6, with 95% confidence intervals plotted as error bars. As the CIs are tight, there was no need to perform additional measurements.

Algorithm 1 Finding the largest n such that the simulation finishes within 10 minutes (i.e. 600 seconds).

procedure FIND n
  $n \leftarrow 1$ ▷ Result to return, iteratively becomes better
  $t \leftarrow 0$
  while $t < 60$ **do** ▷ Ensure that n is large enough before estimating a
    $t \leftarrow$ measure the execution time of CA in seconds with grid length n
    $n \leftarrow 2n$
  while not $540 < t < 660$ **do** ▷ Keep estimating n unless t is within 1 minute of 10 minutes
    $a \leftarrow t/n^2$
    $n \leftarrow b\sqrt{600/ac}$ ▷ Best estimate of n using a
    $t \leftarrow$ measure the execution time of CA in seconds with grid length n
  return n

Apart from the Moonchai model falling slightly short (due to its two-compartment design, effectively doubling the computational load), we were able to simulate over a billion cells in under 10 minutes for all the remaining models with all optimizations enabled. Furthermore, turning any optimization on is always beneficial, except in the case where only optimization (5) is turned on. Although the averages suggest a performance decrease, a two-sample t-test ($\alpha = 0.05$) shows that differences are not statistically significant.

Comparisons *within* the same model shows the effectiveness and generality of the proposed optimizations. By comparing *across* models, we observe differences which are primarily due to two reasons. First, a model such as Rana takes longer than the dos Santos model because of its transition rules: they require more random numbers need to be generated for a given transition rule, and the cell states that require neighborhood checks (i.e., healthy cells). Second, being treatment models, Gonzalez and Rana models maintain higher levels of healthy cells throughout the simulation compared to others (Figure 7). Because healthy cells have transition rules involving neighborhood checks, having more healthy cells leads to more computationally expensive updates.

5 DISCUSSION AND CONCLUSION

Cellular automata have often been developed and implemented by research groups for which "the simplicity of usage was given higher priority than performance" (Guan and Clarke 2010). Simple implementations in popular languages such as Python can run sufficiently well on professional workstations when three conditions are met (Giabbanelli 2019): there is a low number of cells, few update steps, and a small number of repeats. However, in practice, all three conditions may be violated. Using the example of the Human Immunodeficiency Virus (HIV), there is an enormous number of cells (in the order of 10^{11}) (Zhang et al. 1998; Wong and Yukl 2016), a large number of updates (events of interest happen over hours but the lifecourse of the virus is over years) (Murray et al. 2011), and many replications are necessary given that large variations are one of the hallmarks of HIV (Wu et al. 2005). As simple implementations cannot cope with this situation, a vast number of models have been developed (Precharattana et al. 2010; Rana et al. 2015; Golpayegani et al. 2017; Hillmann et al. 2017; Hillmann et al. 2020; dos Santos and Coutinho 2001; González et al. 2013; Jafelice et al. 2015) by making problematic compromises such as artificially 'slowing down' the virus so that it does not spread too fast in a small space. The value proposition of these models is negatively affected: since they cannot cope with simulating baseline dynamics at the appropriate scale, they are also far from supporting features of interest for clinical decision making such as tracking viral mutations and ultimately prediction when/how to change drug regimen. There is thus a necessity to fill the gap between the simplicity of model development and the reliance on professional workstations, as they are clear preferences and common practices from user groups, while improving performances.

We started to address this gap in 2020, through our study on performance bottlenecks (Köster et al. 2020) and our tailored solution for a single HIV CA model (Giabbanelli et al. 2020). In this paper, we

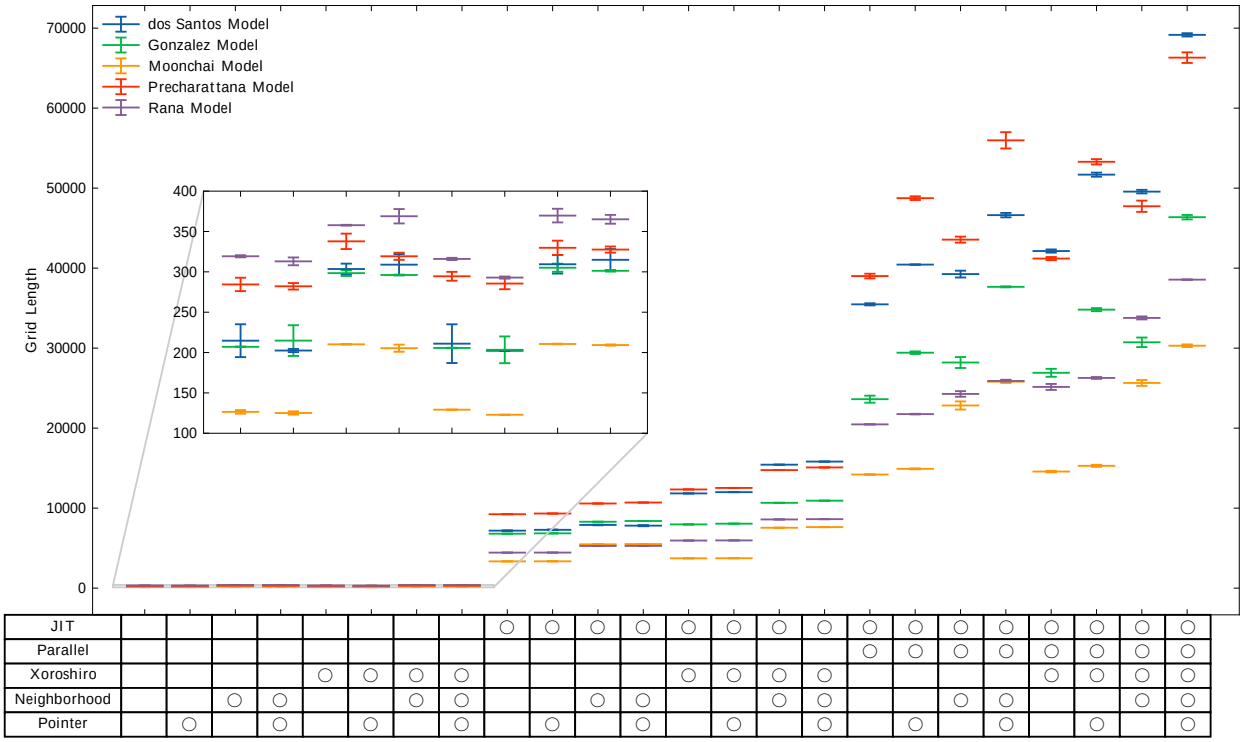

Figure 6: Maximum grid length such that the CA simulation finishes within 10 minutes for all HIV models and all the differently optimized scripts. Error bars plot the 95% confidence intervals. The symbol " " indicates that optimization is active. *The high-resolution figure in the digital version of this article can be zoomed in.*

significantly extended our previous efforts by optimizing the performances of five HIV CA models and contrasting their runtime based on model features. Using our approach and template generator, modelers can move from very small $n \quad n$ grids with n in the order of hundreds to large grids with n in tens of thousands, while continuing to rely on a user-friendly programming environment and accessible hardware. The performance gains depend on model design, as some can run almost five billion cells while others are closer to one billion. Treatment models are at a disadvantage for performance, because their higher prevalence of healthy cells triggers more computationally expensive rules. Future research should focus on optimizing treatment models, since most HIV CA models currently developed reflect the fact that patients would have some level of access to treatment even if adherence is imperfect. As our results show that HIV CA models can now cope with a large number of cells, the next frontier would be to incorporate mutations and drug resistance. These aspects are essential in the biology of HIV as well as in the nature (i.e., which drugs to use) and timing of treatment. While mutations and drug resistance could not be incorporated previously due to computational limitations, the performance gains in our study suggest the feasibility of adding established mutation databases to future simulations, such as the Stanford HIVdb Program (publicly available at https://hivdb.stanford.edu/) which can be accessed during a simulation via a web service.

REFERENCES

Blackman, D., and S. Vigna. 2018. "Scrambled Linear Pseudorandom Number Generators". *arXiv preprint arXiv:1805.01407*.

Delorme, M., and J. Mazoyer. 2013. *Cellular Automata: A Parallel Model*. Dordrecht: Springer Netherlands.

dos Santos, R. M. Z., and S. Coutinho. 2001. "Dynamics of HIV Infection: A Cellular Automata Approach". *Physical review letters* 87(16):168102.

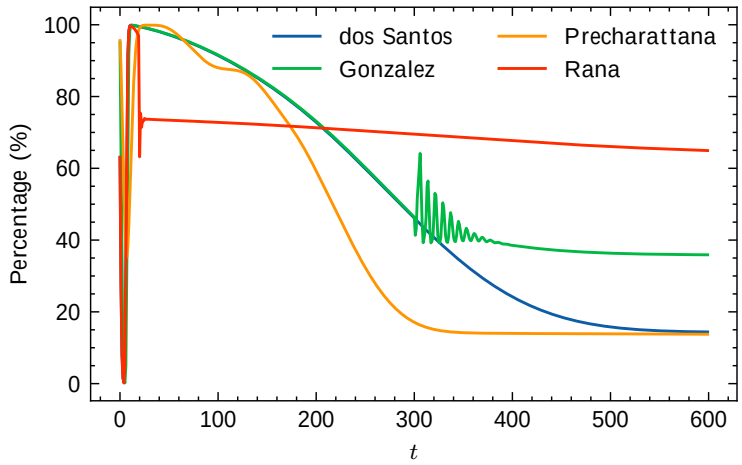

Figure 7: Percentages of cells that involve neighborhood transition rules for different models.

Ghosh, S., and S. Bhattacharya. 2020. "A Data-Driven Understanding of COVID-19 Dynamics Using Sequential Genetic Algorithm Based Probabilistic Cellular Automata". *Applied Soft Computing* 96:106692.

Giabbanelli, P. J. 2012, May. "Ingredients for Student-Centered Learning in Undergraduate Computing Science Courses". In *Proceedings of the Seventeenth Western Canadian Conference on Computing Education*, 7–11.

Giabbanelli, P. J. 2019, Dec. "Solving Challenges at the Interface of Simulation and Big Data Using Machine Learning". In *2019 Winter Simulation Conference (WSC)*, edited by N. Mustafee, K.-H. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, 572–583. IEEE: IEEE.

Giabbanelli, P. J., J. A. Devita, T. Köster, and J. A. Kohrt. 2020, June. "Optimizing Discrete Simulations of the Spread of HIV-1 to Handle Billions of Cells on a Workstation". In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 67–78.

Giabbanelli, P. J., C. Freeman, J. A. Devita, N. Rosso, and Z. L. Brumme. 2019, June. "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.

Giabbanelli, P. J., and P. J. Jackson. 2021, June. "How Do Teams of Novice Modelers Choose an Approach? An Iterated, Repeated Experiment in a First-Year Modeling Course". In *International Conference on Computational Science*, 661–674. Springer.

Giabbanelli, P. J., and V. K. Mago. 2016, June. "Teaching Computational Modeling in the Data Science Era". *Procedia Computer Science* 80:1968–1977.

Gibson, M. J., E. C. Keedwell, and D. A. Savić. 2015, March. "An Investigation of the Efficient Implementation of Cellular Automata on Multi-Core CPU and GPU Hardware". *Journal of Parallel and Distributed Computing* 77:11–25.

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* 392(19):4701–4716.

Gosper, R. 1984, January. "Exploiting Regularities in Large Cellular Spaces". *Physica D: Nonlinear Phenomena* 10(1-2):75–80.

Guan, Q., and K. C. Clarke. 2010. "A General-Purpose Parallel Raster Processing Programming Library Test Application Using a Geographic Cellular Automata Model". *International Journal of Geographical Information Science* 24(5):695–722.

Guan, Q., X. Shi, M. Huang, and C. Lai. 2016, March. "A Hybrid Parallel Cellular Automata Model for Urban Growth Simulation over GPU/CPU Heterogeneous Architectures". *International Journal of Geographical Information Science* 30(3):494–514.

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.

Hillmann, A., M. Crane, and H. J. Ruskin. 2020. "Assessing the Impact of HIV Treatment Interruptions Using Stochastic Cellular Automata". *Journal of theoretical biology* 502:110376.

Jafelice, R. M., C. A. Silva, L. C. Barros, and R. C. Bassanezi. 2015. "A Fuzzy Delay Approach for HIV Dynamics Using a Cellular Automaton". *Journal of Applied Mathematics* 2015.

Köster, T., P. J. Giabbanelli, and A. Uhrmacher. 2020, Dec. "Performance and Soundness of Simulation: A Case Study Based on a Cellular Automaton for in-Body Spread of HIV". In *2020 Winter Simulation Conference (WSC)*, edited by K.-H. Bae, S. Lazarova-Molnar, Z. Zheng, B. Feng, and S. Kim, 2281–2292. IEEE: IEEE.

Moonchai, S., and Y. Lenbury. 2011, March. "Double Compartment CA Simulation of Drug Treatments Inhibiting HIV Growth and Replication at Various Stages of Life Cycle". *International Journal of Mathematics and Computers in Simulation* 5(3):232–241.

Murray, J. M., A. D. Kelleher, and D. A. Cooper. 2011. "Timing of the Components of the HIV Life Cycle in Productively Infected CD4+ T Cells in a Population of HIV-Infected Individuals". *Journal of virology* 85(20):10798–10805.

Ngo-Giang-Huong, N., and A. F. Aghokeng. 2019. "HIV Drug Resistance in Resource-Limited Countries: Threat for HIV Elimination". *EClinicalMedicine* 9:3–4.

Ortigoza, G., F. Brauer, and I. Neri. 2020. "Modelling and Simulating Chikungunya Spread with an Unstructured Triangular Cellular Automata". *Infectious Disease Modelling* 5:197–220.

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, June. "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.

Sengupta, S., and R. F. Siliciano. 2018. "Targeting the Latent Reservoir for HIV-1". *Immunity* 48(5):872–895.

Staubitz, T., R. Teusner, C. Meinel, and N. Prakash. 2016, Dec. "Cellular Automata as Basis for Programming Exercises in a MOOC on Test Driven Development". In *2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 374–380. IEEE.

Vendome, C., D. M. Rao, and P. J. Giabbanelli. 2020, May. "How Do Modelers Code Artificial Societies? Investigating Practices and Quality of Netlogo Codes from Large Repositories". In *2020 Spring Simulation Conference (SpringSim)*, 1–12. IEEE.

Willem, L., F. Verelst, J. Bilcke, N. Hens, and P. Beutels. 2017. "Lessons from a Decade of Individual-Based Models for Infectious Disease Transmission: a Systematic Review (2006-2015)". *BMC infectious diseases* 17(1):612.

Wong, J. K., and S. A. Yukl. 2016. "Tissue Reservoirs of HIV". *Current Opinion in HIV and AIDS* 11(4):362.

Wu, H., Y. Huang, E. P. Acosta, S. L. Rosenkranz, D. R. Kuritzkes, J. J. Eron, A. S. Perelson, and J. G. Gerber. 2005. "Modeling Long-Term HIV Dynamics and Antiretroviral Response: Effects of Drug Potency, Pharmacokinetics, Adherence, and Drug Resistance". *JAIDS Journal of Acquired Immune Deficiency Syndromes* 39(3):272–283.

Zhang, Z.-Q. et al. 1998. "Kinetics of CD4+ T Cell Repopulation of Lymphoid Tissues After Treatment of HIV-1 Infection". *Proceedings of the National Academy of Sciences* 95(3):1154–1159.

Zorzenon dos Santos, R. M., and S. Coutinho. 2001, September. "Dynamics of HIV Infection: A Cellular Automata Approach". *Physical Review Letters* 87(16):168102.

AUTHOR BIOGRAPHIES

JUNJIANG LI is an undergraduate student in the Department of Computer Science and Software Engineering at Miami University, with interests in machine learning and artificial intelligence. His email address is lij111@miamioh.edu.

TILL KÖSTER is a research assistant 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). He holds a PhD from Simon Fraser University and taught at several nationally ranked American universities. His research interests include network science, machine learning, and simulation and applied to human health behaviors. His email address is giabbapj@miamioh.edu. His website is https://www.dachb.com.