SIMCoV-GPU: Accelerating an Agent-Based Model for Exascale

Kirtus G. Leyba
Arizona State University
Tempe, United States
kleyba@asu.edu

Steven Hofmeyr
Lawrence Berkeley National Laboratory
Berkeley, United States
shofmeyr@lbl.gov

Judy Cannon
University of New Mexico
Albuquerque, United States
JuCannon@salud.unm.edu

Melanie Moses
University of New Mexico
Albuquerque, United States
Melaniem@cs.unm.edu

Stephanie Forrest
Arizona State University
Tempe, United States
steph@asu.edu

ABSTRACT

Modern supercomputers rely on *graphics processing units* (GPUs) to achieve unprecedented computational capabilities. Multi-node computation with GPUs promises to accelerate and scale simulations dramatically across many domains, and many scientific simulations have been adapted to this new paradigm of supercomputing. However, agent-based models (ABMs) are a class of simulations that to date have seen little development for multinode, multi-GPU supercomputers because their computation flow poses unique algorithmic and communication challenges for effective performance on GPU enabled supercomputers. In particular, many ABMs have irregular and dynamic communication patterns, resource competition that causes race conditions, and unpredictable effects on load balancing. We studied the *Spatial Immune Model of Coronavirus*, or SIMCoV, as a target ABM application for acceleration. SIMCoV is a large-scale ABM which simulates the spread of viral infection through the epithelial tissue of the lungs and models the immune response with diffusing inflammatory signals and mobile T cell agents. Our multinode, multi-GPU implementation of SIMCoV achieves significant speedups over a competitive baseline version, up to 11.9x with a ratio of 32 CPU cores to a single GPU. The paper describes SIMCoV's GPU-specific optimizations, reports empirical results, and demonstrates effective solutions to the challenges of accelerating ABMs on modern supercomputers.

ACM Reference Format:
Kirtus G. Leyba, Steven Hofmeyr, Judy Cannon, Melanie Moses, and Stephanie Forrest. 2024. SIMCoV-GPU: Accelerating an Agent-Based Model for Exascale. In *International Symposium on High-Performance Parallel and Distributed Computing (HPDC '24), June 3–7, 2024, Pisa, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3625549.3658692

HPDC '24, June 3–7, 2024, Pisa, Italy

ACM ISBN 979-8-4007-0413-0/24/06…$15.00
https://doi.org/10.1145/3625549.3658692

1 INTRODUCTION

Computational models of real-world complex systems are found across diverse scientific domains. Frequently, researchers are interested in adaptive systems with many moving parts that feature complex interactions, decision making, and communication. For example, researchers have modeled the impact of different management strategies on forest fires [30], the flow of traffic in road topologies [29], and the economic dynamics of demand shocks impacting a supply chain [24]. These models are all categorized as *Agent-Based Models* (ABMs). ABMs model agents that live in a spatial environment, make decisions and interact with one another and the environment. ABMs are an essential tool for researchers because they can encode expert knowledge about components and interactions within a system and then study its large-scale emergent dynamics. In many cases, however, the utility of an ABM is limited because it is too computationally expensive to execute at scale, either because too many agents are required or communication between computational units is prohibitively costly.

To improve their scaling and manage highly complex models, ABMs are frequently parallelized, but they have rarely been deployed at scale on multinode, multi-GPU systems. When ABMs are implemented on GPUs, they tend to run on a single GPU, a single node, or a single GPU per node [3, 28]. These limitations are significant, because the current generation of supercomputers relies on clusters of GPU nodes to drive their immense computational capabilities. The supercomputer Frontier, for instance, achieved 1.1 exaflops in 2022 by relying on both CPU and GPU computation [1]. Realizing the potential of *exascale* supercomputers to run large ABMs is important: first, it can increase the size of ABMs that can be executed (crucial for our application), second, it can increase the number of simulations that can be performed (important for exploring parameter regimes and repeating highly stochastic simulations), and third, it can support models of increased complexity to provide more salient results and better predictive capabilities.

In this work, we focus on a single ABM, SIMCoV [25], which illustrates the need to run at scale and highlights many of the complexities associated with implementing multinode, multi-GPU ABMs. SIMCoV (*Spatial Immune Model of Coronavirus*) is a high-performance computing (HPC) application that simulates a 2D or 3D voxel world of epithelial (tissue) cells that is used to spatially

model the human lung, which has billions of individual cells. SIMCoV is an exemplar for multinode, multi-GPU acceleration because of its large scale and dynamic computational landscape. SIMCoV features mobile agents that compete for limited space, introducing communication requirements to resolve the competition, as well as diffusing concentrations from multiple initial locations, causing the expensive to compute simulation regions to grow and vary over time and between simulations. SIMCoV shares these computational challenges with other ABMs, which further motivates our endeavour to accelerate it. Throughout this paper we refer to our multinode, multi-GPU update to SIMCoV as *SIMCoV-GPU*, which we distinguish from the baseline implementation, referred to as *SIMCoV-CPU*. We focused on a specific HPC application, but are optimistic that our findings will provide helpful insight for researchers seeking to develop other many-GPU simulations, particularly ABMs where mobile agents induce expensive communication and dynamic load balance issues.

The contributions of our work are as follows:

- A multinode, multi-GPU implementation of a large complex ABM. Our development includes applying GPU and cluster computing focused optimizations to address costly communication, mobile agents that compete for spatial resources, and unpredictable load-balancing.
- An empirical evaluation of performance improvement, demonstrating 11.9x speedup over a competitive baseline implementation on a system with a ratio of 32 CPU cores to one GPU, which ideally could provide at most 15.6x speedups.
- Generality: ABMs are a class of computational models that have yet to take advantage of state-of-the-art supercomputers. The approaches outlined here present a road map for the deployment of large-scale ABMs on exascale, multinode, multi-GPU supercomputers.

The remainder of the paper is organized as follows:

- Section 2 provides details of SIMCoV and relevant background information necessary for accelerating ABMs with multiple GPUs.
- Section 3 describes our approach to developing SIMCoV-GPU and where it necessarily differs from SIMCoV-CPU.
- Section 4 reports on an empirical evaluation of SIMCoV-GPU's performance and compares it to a competitive baseline.
- Section 5 discusses relevant prior work, Section 6 discusses our findings, highlights limitations and suggests future work, and
- Section 7 concludes the paper.

2 BACKGROUND

This section briefly reviews earlier efforts to accelerate ABMs and then describes the baseline implementation of our target application, SIMCoV-CPU and why it is an interesting target for many-GPU acceleration.

2.1 Parallel Agent-Based Models

ABMs, which are sometimes referred to as individual-based models [14], are computational models that simulate large dynamic systems by describing the behavior and interactions of their individual actors, referred to as agents. ABMs are specified by a collection of agent types, rules for how agents interact with each other and their environment (behavior), and a topology that constrains their movement in the environment and which agents they interact with. Agent rules can be static or adaptive; in some cases, all agents have identical rules and in others, each agent has its own set of rules which it can update over time through learning or evolution. For example, in a cancer simulation, different agent types could correspond to different types of cells in the body; each individual cell might behave differently, depending on how many mutations it has suffered; and cell movement could be constrained to a three-dimensional local neighborhood, representing nearby physical space [16]. Once the component parts of an ABM are specified, the system can be studied, much like in a laboratory setting, by running experiments with varied parameters and setups.

This bottom-up modeling approach has the advantage that the simulations can reveal interesting emergent behaviors, which are not predictable ahead of time, despite relatively simple agent rules. In cases where a certain high-level behavior has been observed, a plausible underlying mechanism can be discovered by testing different ABM configurations to see which ones most closely mimic the observed phenomenon. ABMs have been used to study a wide variety of complex systems including those in environmental sciences [30], social sciences [5–7], biological sciences [10], and other domains. ABMs can divide their computation into discrete time steps or discrete events [4]. In both cases, ABMs can be parallelized by distributing the operations of agents to computational units (threads, processes, GPUs, etc.).

Parallelization is necessary for ABMs when they are used to simulate large populations of agents with many interacting components. The emergent properties of ABMs can easily differ based on population size, and it is often a worthwhile research question itself to understand the impact of population scale on an ABM. In the example we describe below, the environment is organized as a volume of voxels, each five microns cubed, so billions of cells must be modeled to achieve realistic scale simulations consistent with the number of cells in a human lung. For large complex systems, the ability to simulate efficiently at scale is essential for relating model results to the real world. As mentioned earlier, this is computationally challenging due to the sheer number of agents, dynamic workloads, and the communication overhead that arises from managing agent interactions among themselves and their environment.

Each ABM agent has a location on a network topology which is defined for the specific problem being studied. Nearby agents may compete for some resource, e.g., two agents may both attempt to move to the same spatial location in a discrete grid [27]. In a serial implementation, such conflicts are resolved by random tiebreak. In a parallel environment, however, extra care must be taken to ensure that all relevant processes agree on the simulation state after tiebreaks. This can be solved using random tiebreaks and communicating the result between the processes, but this communication can be particularly costly on multinode, multi-GPU systems.

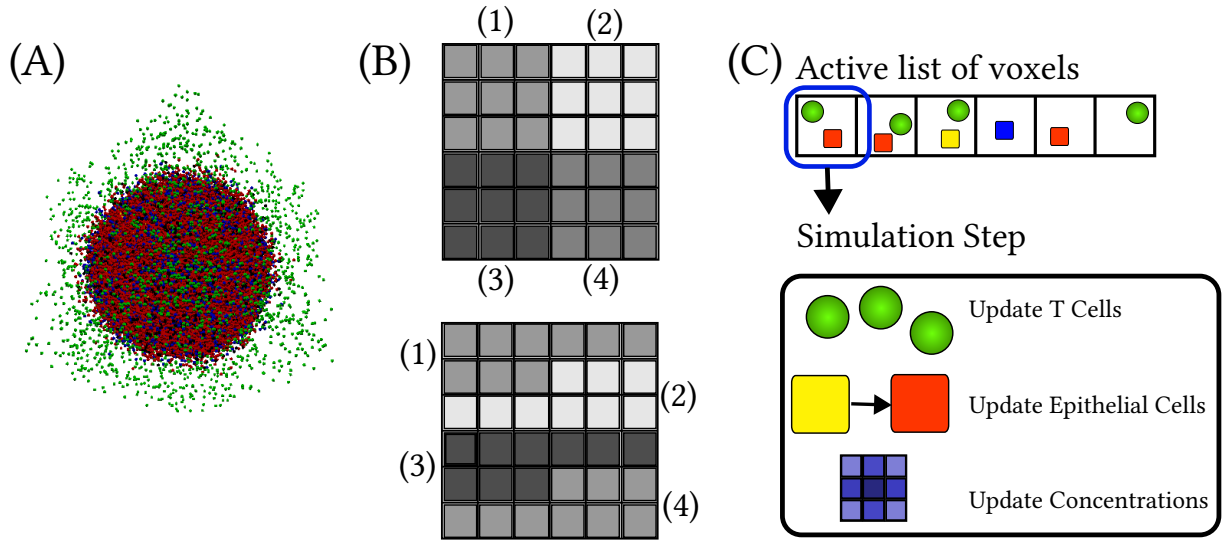

Figure 1: A diagram of the behavior of SIMCoV. (A) A 3D visualization of a small SIMCoV simulation seeded with a single FOI. The spreading damage of the infection is shown. Apoptotic (red) and expressing (blue) epithelial cells are seen on the boundary of the growing infection. The T cells (green) randomly walk searching for infected cells to bind to. (B) The simulation is distributed to processes via either block (top) or linear (bottom) domain decomposition, which has impacts on communication overhead. The processes are labeled 1 through 4 in this 2D example. (C) For a single simulation step, each process iterates over its portion of the *active list* of voxels and performs updates: Moving and binding for T cells, updating epithelial cell states, and diffusing concentrations.

2.2 SIMCoV

SIMCoV was developed as an ABM to simulate the spread of a viral infection (parameterized for SARS-CoV-2) throughout the lung tissue of a host as well as a corresponding immune response. SIMCoV differs from earlier within-host infection models by including spatial structure. These earlier models focus on the aggregate statistics of an infection over time using ordinary differential equations (ODEs) [18, 31]. In the earlier work, ODEs model populations of cells, virus, and other entities as being well-mixed, which means that all possible interactions between the entities are equally likely, regardless of where the entities are located. However, because many biological processes are localized to nearby cells, spatial effects can be important. Examples in SIMCoV include the impact of how the initial infection is distributed throughout the tissue and the spatial distribution of immune cells that can fight the infection. In particular, SIMCoV can model varying numbers of FOI (foci of infection).

The spatial environment of a SIMCoV simulation is a 2D or 3D grid of voxels, and the hierarchical branching structure of the lung can be overlaid on either the full 3D volume or a single slice represented by a 2D grid. Each voxel can be empty or contain model components, including epithelial cells, immune cells, and molecular concentrations. Earlier results showed that SIMCoV can match longitudinal patient data (spread of virus during an infection) by fitting three key parameters of the simulation—parameters that are expected to vary across individuals [25].

A detailed description of the SIMCoV model appears in [25], and figure 1 illustrates the most relevant aspects of the model for the high-performance computing (HPC) implementations. SIMCoV's agents consist of epithelial cells, which are stationary but can be in multiple states (healthy, incubating, expressing, apoptotic, or dead) and motile T cells that can trigger a process known as apoptosis that kills infected epithelial cells. Epithelial cells transition stochastically from an incubating state (producing virus while not being detectable to T cells) to an expressing state (now detectable to T cells) according to an incubation period which each epithelial cell draws from a parameterized Poisson distribution. In the absence of stimulation, T cells circulate through the vascular network of the body, which is modeled implicitly as an available pool of T cells. A high concentration of inflammatory signal triggers T cells to leave the vasculature near infected regions of the simulation, and when they encounter infected epithelial cells they bind to them and initiate the death of the infected cell (apoptosis). A single voxel contains a maximum of one T cell and one epithelial cell at any point in time. Structure is defined for the simulation, such as branching airways in the lung, by leaving some voxels empty without epithelial cells, although in this paper we evaluate 2D simulations which correspond to the simulations that fit best to patient data in [25].

In addition to discrete agents, where each instance is represented explicitly in the simulation, the SIMCoV environment contains two continuous quantities, which vary over time and affect the behavior of agents: the virus itself and an inflammatory signal that indicates the presence of infection. These are represented as concentrations that diffuse through simulation space according to parameterized diffusion rates. The virus infects epithelial cells as it spreads through the simulation space, while the inflammatory signal increases the chances of T cells extravasating at locations with high concentrations. Each time a T cell has a chance to extravasate, a voxel in the tissue is selected uniformly at random, and with a probability

proportional to the concentration of the inflammatory signal at that voxel, the T cell will enter the simulation at that location.

The original parallelization strategy of SIMCoV divided the simulation domain into subsets of voxels and distributed them to individual CPU processes. The multiprocessing framework for SIMCoV is UPC++ [8], which is a parallel computation library that uses a Partitioned Global Address Space (PGAS) to enable interprocess communication. UPC++ includes features such as remote procedure calls (RPCs) and support for direct GPU-to-GPU communication.

SIMCoV-CPU can be parallelized by subdividing the simulation space into subdomains using either linear, 2D, or 3D domain decomposition. Some of the simulation work in SIMCoV can be performed locally, such as virus infecting an epithelial cell. Other operations require communication, such as moving T cells across process boundaries. These communication instances are handled in the CPU version of SIMCoV using RPCs in UPC++ which allow one process to asynchronously queue the execution of a function on another process.

This approach is convenient for CPU-based parallelism and reduces the development effort required to implement the model behaviors, but it is not efficient for exploiting the massive parallelism available in GPUs. A multinode, multi-GPU SIMCoV should process as much simulated space as possible within each GPU kernel and minimize costly communication and other general purpose GPU programming pitfalls. Necessary changes to the implementation to create SIMCoV-GPU are described next in section 3.

3 ADAPTING SIMCOV TO GPUS

SIMCoV-GPU is implemented using NVIDIA's Compute Unified Device Architecture (CUDA) for GPU kernels and GPU library calls, and relies on UPC++ to handle interprocess communication and GPU-to-GPU copies. Implementing a computationally efficient version of SIMCoV to run across multiple GPUs required substantial changes to algorithms and data structures. SIMCoV-CPU was designed to allow for communication within simulation iterations, such as T cells binding to epithelial cells across process boundaries. Additionally, it relies on dynamic data structures and the global address space provided by UPC++. These tools are either not available within GPU kernels or would introduce prohibitive overhead when implemented on GPUs. SIMCoV-CPU also reduces the computational work on inactive regions by tracking the active voxels in an active list. Maintaining a dynamic data structure that is shared between GPUs would also incur a high communication cost. These new requirements for SIMCoV-GPU motivated us to modify the T cell agent algorithm, introduce memory tiling to manage inactive regions, and take advantage of GPU shared memory to reduce global memory accesses and atomic operations.

3.1 T Cell Algorithm

SIMCoV models Cytotoxic CD8 T cells explicitly as agents. SIMCoV T cells move randomly from voxel to voxel and bind to any virally infected epithelial cells they encounter, triggering programmed cell death (apoptosis). In SIMCoV-CPU, this behavior is accomplished by iterating over active voxels and executing movement and state updates for the T cells found in those voxels. Since two T cells cannot occupy the same voxel, collisions are resolved using RPCs; a

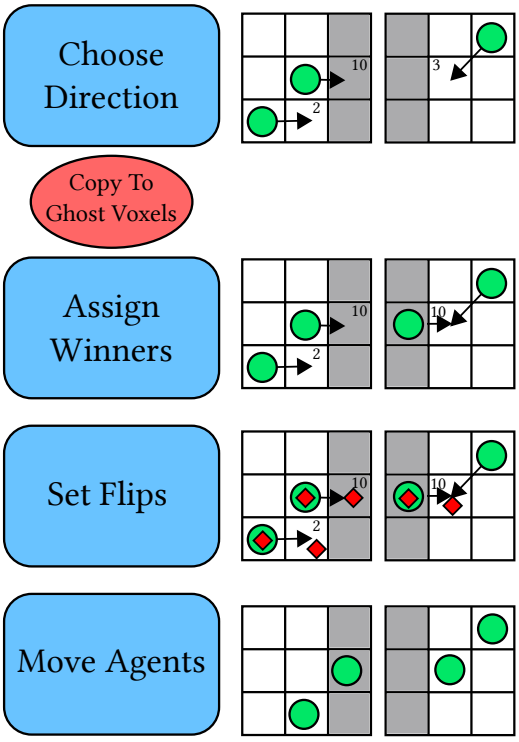

Figure 2: T cell movement algorithm in SIMCoV-GPU. Each T cell chooses a random direction to try to move and stores a randomly generated value at that location. After a copy operation, tie breaks are resolved, voxels are set to flip the presence of T cells on or off depending on if a T cell is entering or leaving (red diamond). Next, the moves are executed. (Left) Blue rectangles indicate the order of kernels, and red ovals indicate copies between GPUs. (Right) Diagram of the memory layout for two adjacent GPUs, illustrating a representative sequence of T cell operations. Gray regions indicate ghost voxels.

T cell is prevented from moving if it tries to move into an occupied voxel. A T cell can also fail to move if a different T cell wins a tiebreak and to move into an unoccupied voxel. T cell binding is treated similarly: multiple T cells attempting to bind the same epithelial cell in one timestep resolve the resource competition in the same way.

SIMCoV-GPU cannot use this approach because within a single simulation timestep, one GPU cannot query the simulation state of a particular voxel. Instead, GPUs acquire information about their boundaries during communication waves placed between computation. During communication, the memory of a GPU's boundary is copied to its neighbors, and stored there in a halo of ghost voxels that surrounds the simulation space. For epithelial cells which don't move and do not interact directly with their neighbors and for concentrations that diffuse according to a stencil code, this is sufficient to compute all updates for a timestep. T cells are more interesting, because they make independent pseudo-random choices of where to move and where to bind every timestep. Simply knowing there

is a T cell on the boundary of a GPU's neighbor in simulation space isn't sufficient to know where that T cell is going to go, and even worse there is no guarantee that a T cell completely unseen to one GPU moves into a location that a local T cell has chosen as its target. This problem arises due to the decision making nature of T cells, and is a particularity of ABMs that is not shared with many other HPC applications. While our T cells are random walkers, a similar spatial resource competition challenge would arise in other ABMs with adaptive agent behaviors such as evolved decision making.

One solution to this challenge is to first communicate the intent of every T cell (where it chooses to move or to bind), perform a communication call, resolve tiebreaks from T cells attempting to move or bind to the same location, and then copy the results back to let each GPU know what to do with their boundary T cells. Fortunately, we can do better and avoid the second communication call. Our approach is similar to that described in [27], where each agent bids on the spatial resource that it is competing for. Our T cell behavior is specialized, however, for the functionality of SIMCoV because the preference for winners is random on each timestep and T cells are not expected to try to move multiple times in any iteration, i.e., T cells can and do run into each other. These adaptations are necessary to match the behavior of SIMCoV-CPU.

The T cell movement algorithm of SIMCoV-GPU is visualized in figure 2. First, the T cells within the sub-domain of each GPU select a random neighbor voxel as a target. Second, each T cell generates a pseudo random number from a large range of integers (true ties where multiple conflicting T cells draw the same value are possible but so unlikely that it is efficient and practical to ignore them). They store that value at their own voxel, and if it is larger than the current value at their target voxel they also store it at the target. Now, a wave of communication between neighboring GPU processes allows each voxel to determine the winning T cell which may or may not be stored in local memory. Finally, T cells that win the tiebreak (or had no competition) are moved in a local computation step on each GPU where they are copied to their destination voxels and erased from their source. Conveniently, if a T cell has moved into the memory space of a GPU, that T cell can safely be instantiated in memory without fear of duplication. Since the tiebreak is deterministic in this regard, the GPU from which the T cell came will erase it.

In the subsequent subsections 3.2 and 3.3 we introduce two GPU focused optimizations used in SIMCoV-GPU.

3.2 Memory Tiling

There are many possible states that a SIMCoV simulation could exist in, and the amount of actual activity in the simulated space can vary drastically over time. For instance, a region of lung tissue that does not have any virus or inflammatory signal does not change from one timestep to the next. It is important to take advantage of this to rapidly compute timesteps with low activity and not waste computational resources with poor load-balancing. SIMCoV-CPU addresses this with an *active-list* data structure to track which voxels can possibly be changed each timestep. Each process in SIMCoV-CPU maintains its own active list, and it is straightforward to determine which voxels could possibly change from one timestep to the next. This is true over process boundaries as well, because

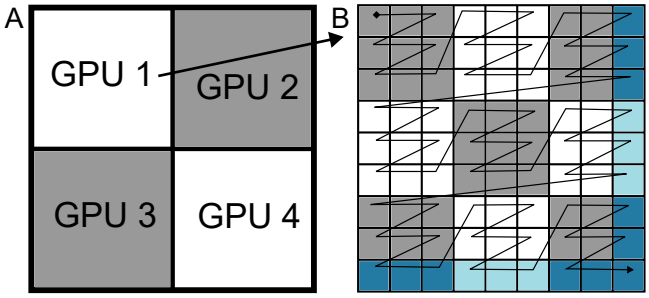

Figure 3: The memory layout of a SIMCoV-GPU run. This example is a 2D simulation on four GPUs. (A) The GPUs subdivide the simulation domain equally using 2D domain decomposition. (3D domain decomposition is applied similarly for 3D simulations). (B) A zoomed in view of GPU 1's voxel layout in memory. In this example, 3x3 memory tiles are used, indicated by lighter and darker regions. Blue voxels belong to the ghost halo. The order of voxels in memory is shown by the zig-zag path that traverses the space.

when a process uses an RPC to communicate with its neighbor, that RPC can add the affected voxels to the active-list. As previously explained, in SIMCoV-GPU each GPU performs updates over its space and communicates its boundaries in separate steps, so the active-list would not be updated across process boundaries unless this was handled during communication. An additional challenge is that the memory footprint of tracking the active-list would have to be proportional to the number of voxels or the active-list would have to be a dynamic data structure which could at times be as large as the total number of voxels anyway. Either approach is unattractive for GPUs due to their restrictive memory constraints.

Instead we use *memory tiling* to accomplish this optimization in SIMCoV-GPU, an approach inspired by optimizations in computer graphics [19]. In SIMCoV-GPU a *tile* is defined as the memory (including epithelial cells, T cells, and concentrations) containing a sub-domain of the simulation space. We require that the dimensions of a tile allow for an integer number of tiles to subdivide the simulation along each dimension. Note that this is a lower order than the domain subdivision that distributes simulation regions to GPUs. Each tile stores its voxels contiguously in memory, which has an added benefit of data locality. For GPUs especially, global reads and writes are expensive, and it is more likely that voxels nearby are cached when using memory tiling. Tiles are tracked as active or inactive and SIMCoV-GPU kernels perform computation only on the active tiles. Tiles are activated when a check is executed via a GPU kernel that sweeps the simulation space looking for activity. To minimize the cost of this update, we choose to only perform the check periodically. The maximum the period for active tile checking can be set is to the size of the side of a tile, on the condition that when a tile activates it also activates a buffer of tiles around it one tile thick. We know this will safely encapsulate all of the simulation activity because nothing in SIMCoV can move faster than one voxel per timestep. We set tiles that contain ghost voxels to active always to ensure that entities that enter from other GPU memory spaces are updated correctly. Importantly, we find that

the overhead of checking tiles is much smaller than the benefit of skipping inactive regions. This suggests that this optimization can be applied in other ABMs including those with fast moving agents or agents that interact over long ranges that would increase the rate of active tile checking. A visualization of the complete domain decomposition of SIMCoV-GPU including memory tiling is shown in figure 3.

3.3 Fast Reduction

SIMCoV collects a variety of statistics during execution, which are used to interpret model output. These include aggregate quantities such as the total count of virus molecules, the total number of T cells in the tissue, and the total number of epithelial cells in each of their possible states. These are collected each time step to enable time series analysis of infection dynamics. In SIMCoV-CPU, each process updates these quantities locally and then a UPC++ directive triggers a reduction, with a single process logging the totals to a file on disk. In SIMCoV-GPU each kernel is executed with thousands of threads, so within a single process there would be memory contention and race conditions when updating global values. One solution is to use atomics to update simulation statistics within kernels. Atomics in GPUs introduce considerable overhead, which gets worse when launch configurations use more threads or larger block sizes. We find, perhaps counterintuitively, that it is considerably faster to perform a reduction over every single voxel in the simulated space than include atomics throughout a single simulation update. We can further enhance the reduction by using a tree-like parallel reduction method to take advantage of device shared memory and reduce the total number of atomic operations [17]. In this reduction, each GPU thread first accumulates values for a subset of the voxels, and then each thread block accumulates the values of its threads, and finally the CPU process that manages each GPU reduces the globals across all GPUs using a UPC++ directive.

These optimizations focused on GPU programming and multinode operation, which is a lens that must be used in order for ABMs to effectively take advantage of modern supercomputers. Next, we investigate the impact these optimizations have on SIMCoV-GPU in subsection 3.4.

3.4 Profiling Optimizations

The optimizations implemented for SIMCoV-GPU focused on several efficiency aspects that are especially important in multinode, multi-GPU programming: load-balancing, global memory access patterns, data locality, and atomic operations. In order to evaluate the performance impact of these optimizations we analyzed four prototypes of SIMCoV-GPU with different stages of optimization. *Unoptimized SIMCoV-GPU* iterates over the entire simulation space each timestep without tracking active regions and and uses atomics to accumulate values within GPU memory, *Fast Reduction SIMCoV-GPU* includes only the tree-like reduction described in section 3.3, *Memory Tiling SIMCoV-GPU* uses the memory tiling described in section 3.2 but forgoes the fast reduction method, and finally *Combined SIMCoV-GPU* uses both optimizations in conjunction. We collected profiling information experimentally on the Arizona State University (ASU) Agave supercomputer using 4 V100 GPUs on a single node. We performed a simulation with dense activity (1024 foci of infection (FOI)) and accumulated the total time in three categories of work: updating agents, updating concentrations, and reducing statistics. The results of this analysis are shown in figure 4.

These results first highlight that reductions take up a very large portion of the computational workload, and are an important target for performance improvement. We also see that both optimizations provide some speedup on their own. Unsurprisingly, the fast reduction approach vastly outperforms the unoptimized version. Also as expected, memory tiling reduces the runtime used on updating agents (concentrations are included in the figure for clarity but do not take considerable time to compute at this scale). It is interesting that memory tiling also improves the performance of reductions, likely due to the enhanced data locality reducing slow memory accesses as the reduction kernel sweeps the simulation space. Finally, we see that the optimizations combine very effectively, which indicates that their speedups come from mostly independent effects.

4 EVALUATION

We conducted an evaluation of the performance of SIMCoV-GPU and compared it to a competitive baseline version of SIMCoV-CPU. First, we conducted a correctness evaluation to verify that SIMCoV-GPU and SIMCoV-CPU compute the same simulation. Next, we evaluated how the two implementations of SIMCoV scale in three ways: *strong scaling*, which highlights the benefit of more compute resources for a static problem size; *weak scaling*, which reveals how additional computational resources enable larger simulation domains; and *scaling foci of infection*, which reveals how an important variable of the simulation affects performance. The configurations of our evaluations are shown in table 1. We chose these problem sizes as a function of our available compute resources. We performed three trials for each configuration, except in the FOI scaling experiment where we only performed a single SIMCoV-CPU trial at 512 FOI and no trials for SIMCoV-CPU at 1024 FOI. This was due to limitations in the computational resources available for this project.

The correctness evaluation was conducted on ASU's Sol supercomputer [20] which features 61 GPU capable nodes, most commonly with 128 CPU cores and four NVIDIA A100s. The scaling experiments were conducted on the NERSC Perlmutter supercomputer [2] which also has 128 CPU cores and four NVIDIA A100s per node.

4.1 Correctness

Ideally, researchers can use either SIMCoV-CPU or SIMCoV-GPU to investigate the same scientific questions without worrying about semantic differences between the implementations. SIMCoV is inherently stochastic because many of its behaviors are generated by pseudo-random number generators (PRNGs). These include which epithelial cells become infected, when a T cell moves to a neighboring voxel, and many more. For modeling purposes, it is not critical that the PRNG implementation in each version be identical—only that they both provide the same long-term results on average.

We also note that the version of SIMCoV-CPU used for this paper was slightly modified from the publicly available version referenced by the original paper. We identified a source of nondeterminism

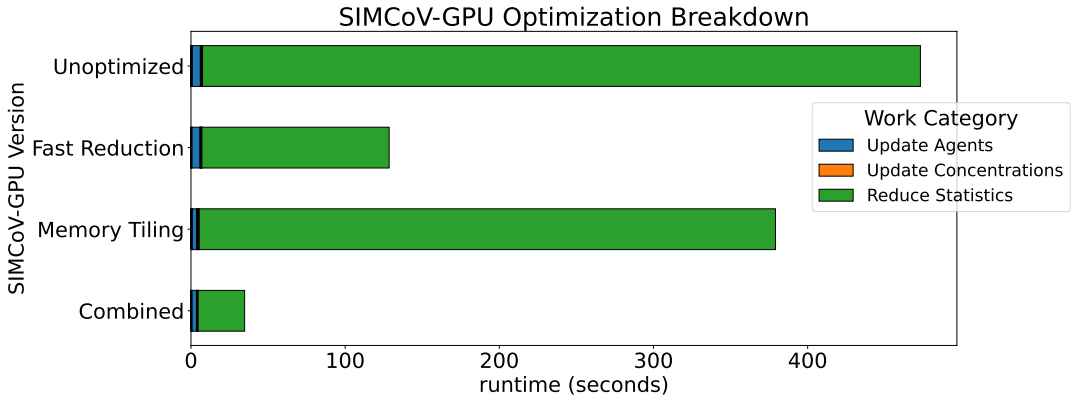

Figure 4: A breakdown of the runtime needed for components of a SIMCoV-GPU simulation broken down by the category of the work being done. Four SIMCoV-GPU versions are plotted: an unoptimized version, a version that only implements fast reductions with shared memory, a version that only implements memory tiling, and finally a version that implements both optimizations.

Experiment	Min. Dimensions	Max. Dimensions	Min. FOI	Max. FOI	Min. {GPUs, CPUs}	Max. {GPUs, CPUs}
Correctness	[10,000x10,000x1]	[10,000x10,000x1]	16	16	{4,128}	{4,128}
Strong Scaling	[10,000x10,000x1]	[10,000x10,000x1]	16	16	**{4,128}**	**{64,2048}**
Weak Scaling	**[10,000x10,000x1]**	**[40,000x40,000x1]**	**16**	**256**	**{4,128}**	**{64,2048}**
FOI Scaling	[20,000x20,000x1]	[20,000x20,000x1]	**64**	**1024***	{16,512}	{16,512}

Table 1: Configuration of the performance evaluation of SIMCoV-GPU versus SIMCoV-CPU. Quantities that vary for an experiment are indicated in bold. All varying quantities double from the previous trial starting from the minimum up to and including the maximum. Computational units are reported as a tuple in brackets: {# of GPUs, # of CPUs}. *We were unable to perform a 1024 FOI trial for SIMCoV-CPU due to computational resource limitations.

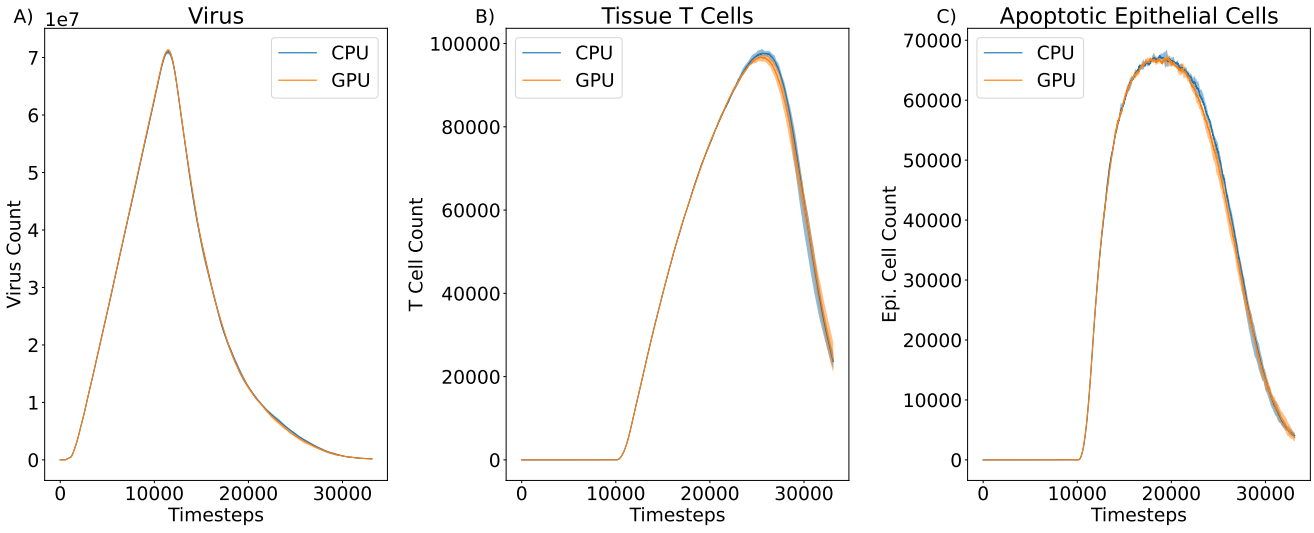

Figure 5: Comparison of aggregate statistics between SIMCoV-CPU (blue) and SIMCoV-GPU (orange) as time-series over the course of a simulated infection. The shaded region shows the minimum and maximum of the statistics across five trials. Plotted are (A) the total count of virus in the simulation, (B) the total count of T cells within the tissue, and (C) the total count of apoptotic epithelial cells.

Stat (Peak)	Pct. Agree.	CPU STD	GPU STD
Virus	99.68	3.1×10^5	2.2×10^5
T cells	99.01	715.82	648.05
Apop. Epi. Cells	99.42	201.09	355.81

Table 2: SIMCoV-GPU Correctness: We show the percent agreement of simulation statistics at their peak as well as the standard deviations of those statistics across five runs. Presented statistics are peak virus count, peak tissue T cell count, and peak apoptotic epithelial cell count.

related to thread ordering in the original implementation and modified it for our experiments. T cells could sometimes be more mobile than expected, and that behavior depended in part on the number of parallel processes used. We standardized this for SIMCoV-CPU and SIMCoV-GPU by enforcing a staged version of T cell movement in which all the T cells first prepare their moves and bindings in one wave of computation, and then in the next wave they execute the queued behavior. This produces more deterministic behavior overall and is more interpretable than the previous implementation. This modification did not affect the parallelization strategy of SIMCoV-CPU which still uses RPCs to handle tiebreaks.

We measured correctness by comparing SIMCoV-GPU and SIMCoV-CPU across five runs using the same parameter set. Both the SIMCoV-CPU and SIMCoV-GPU evaluations were performed on the ASU Sol supercomputer using 128 CPUs on a single node for SIMCoV-CPU and 4 A100s on a single node for SIMCoV-GPU. We used a 2D simulation of a slice of lung tissue 10,000 by 10,000 voxels in size and ran each trial for 33,120 time steps. This duration equates to a simulated time of approximately 23 days, which covers the most common duration of a SARS-CoV-2 infection (and many others as well).

We find that the long-run behavior of SIMCoV-GPU and SIMCoV-CPU is very close given identical initial conditions. Over several runs, the mean of important statistics, e.g., total quantity of each possible epithelial cell state and total quantity of virus, track closely throughout the simulation (figure 5). The percent difference of the mean values of the most relevant statistics reported by SIMCoV are shown in table 2. The peak quantities of virus and T cells are two particularly important statistics for researchers using SIMCoV in practice and the number of apoptotic cells depends highly on the pseudorandom nature of the simulation, so we chose them to study whether or not the two compute platforms produce similar results. Specifically, no statistic was observed to vary more than one percent between the two simulations over the course of their runs, which is much tighter than overall precision of the model.

4.2 Strong Scaling

For our strong scaling experiment we selected a representative problem size: a 10,000 by 10,000 2D slice of epithelial cells. We chose this base problem configuration because it is approximately the number of voxels (100 million) that fit into the A100s' available memory, and patient CT lung images are organized into multiple 2D slices. The simulation was instantiated with 16 FOI (spatially distinct seeds of the infection) and was executed for 33,120 timesteps.

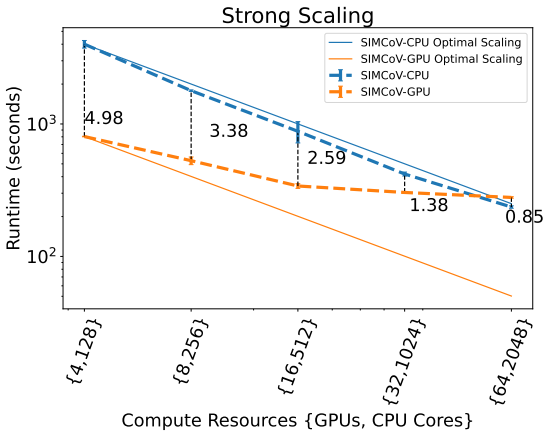

Figure 6: Strong scaling performance of SIMCoV-GPU (orange) versus SIMCoV-CPU (blue). We doubled the number of computational units for each subsequent experiment while maintaining the simulation size throughout. The number of GPUs and CPUs is shown as a tuple in brackets on the x-axis. The runtime of the simulations is plotted on the y-axis in seconds. The plot is on a log-log **scale. The speedup of SIMCoV-GPU over SIMCoV-CPU is shown adjacent to the black dashed-line separating the CPU runs and GPU runs. Optimal scaling is approximated by starting with the mean runtime of the smallest scale experiment, and halving it each subsequent experiment.**

The default COVID-19 parameters from Moses et al. [25] were used. We started with 4 GPUs on a single node for SIMCoV-GPU and 128 CPU cores on a single node for SIMCoV-CPU. For each subsequent trial we doubled the compute resources by doubling the number of nodes. The measured runtime in seconds are reported in figure 6.

The results show that while SIMCoV-GPU significantly outperforms SIMCoV-CPU in the base case, it quickly saturates at this problem size when more computational resources are allocated, as seen by scaling curve deviating from optimal as the number of GPUs increases. This is expected and highlights the limitation of using more GPUs than is appropriate for a given problem size. Unsurprisingly, it is more appropriate to use SIMCoV-GPU on larger problems. A small number of GPUs can still greatly benefit small simulations from their flat performance improvement over the corresponding amount of CPU cores. Such use cases include parameter sweeps and data fitting for small simulations because they require many runs with varied configurations.

4.3 Weak Scaling

Our base instance for weak scaling is identical to that for strong scaling. Unlike strong scaling, however, each doubling of computation resources corresponds with a doubling of the problem size. At each subsequent simulation configuration the FOI is also doubled to fill the new space with proportional amounts of activity. Figure 7 shows the results of this experiment.

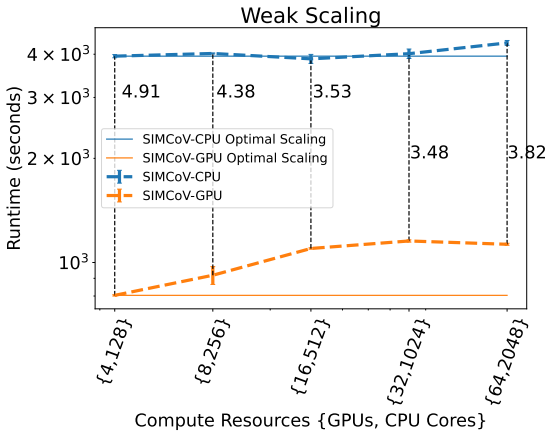

Figure 7: Weak scaling performance of SIMCoV-GPU (orange) versus SIMCoV-CPU (blue). We doubled the number of computational units, the simulation size in voxels, and the number of FOI for each experiment. The number of GPUs and CPUs is shown as a tuple in brackets on the x-axis. The runtime of the simulations is plotted on the y-axis in seconds. The plot is on a log-log **scale. The speedup of SIMCoV-GPU over SIMCoV-CPU is shown adjacent to the black dashed-line separating the CPU runs and GPU runs. Optimal scaling is approximated as remaining constant from the mean of the smallest scale experiment.**

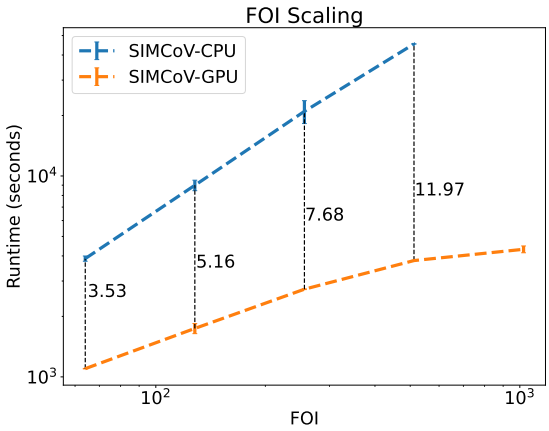

Figure 8: Impact of increased foci of infection (FOI) on the performance of SIMCoV-CPU (blue) versus SIMCoV-GPU (orange). Experiments were performed on four Perlmutter nodes with sixteen GPUs and 512 CPU cores total. We doubled only the number of FOI for each experiment. The number of FOI is shown on the x-axis. The runtime of the simulations is plotted on the y-axis in seconds. The plot is on a log-log **scale. The speedup of SIMCoV-GPU over SIMCoV-CPU is shown adjacent to the black dashed-line separating the CPU runs and GPU runs.**

These results are more favorable to SIMCoV-GPU. We see again that in the early trials SIMCoV-GPU outperforms SIMCoV-CPU. As the problem size and computational units used increase together, there is a higher initial cost of parallelism in SIMCoV-GPU as indicated from the increase in runtime from 4 GPUs to 16. Once this initial increase in runtime is paid, however, SIMCoV-GPU's performance remains nearly constant while SIMCoV-CPU begins to suffer performance loss. This is evidence that SIMCoV-GPU enables larger scale simulations that SIMCoV-CPU, such as the future goal of simulating a full lobe or the full lung.

4.4 Foci of Infection

SIMCoV-GPU and SIMCoV-CPU provide alternative approaches for minimizing the time spent iterating over inactive regions. Additionally the most expensive parts of a SIMCoV execution occur when the simulation space is highly active (a wide distribution of T cells and infection). We compare the performance of SIMCoV-GPU and SIMCoV-CPU under these scenarios by varying the initial conditions of the simulation, which in this case is controlled by the number of initial foci of infection, or FOI. Our FOI experiments were conducted on 4 nodes (16 GPUs, 512 CPU cores) on a 2D slice of epithelial cells with dimensions 20,000 by 20,000. This corresponds to the 4-node weak scaling test. We leave all other parameters identical to the baseline cases and present the results of the FOI experiment in figure 8.

We find that increased FOI is SIMCoV-GPU's strong suit. The GPU implementation maintains sublinear increase in runtime, while the CPU version does not. This sublinear increase in runtime is explained by the fact that as the number of FOI increases the simulation approaches a state of maximum activity. As this threshold is approached, the impact of additional FOI is reduced. We see that SIMCoV-GPU benefits from this threshold at fewer FOI indicating less overhead from massive parallelism versus SIMCoV-CPU. SIMCoV-GPU appears to be highly effective at rapidly completing large simulations with widespread infections.

5 RELATED WORK

Accelerating scientific codes has been an important focus of HPC research throughout the field's history. Recent work that deploys scientific codes on multinode, multi-GPU systems emphasizes models such as particle-in-cell physics simulations or materials simulations. These simulations generally do not face the challenge of local entities in the simulation making arbitrary decisions (such as T cell motion in SIMCoV), which simplifies the handling of communication boundaries. In this work we address that additional complexity with a deterministic tiebreaking step during resource competition. Related approaches to accelerate parallel ABMs use multithreading, multiprocessing, or small counts of GPUs. Examples of work that use multiprocessing, generally with MPI, include e.g., Care HPS [9], the EMEWS framework [26], and Repast HPC [12]. In our work, SIMCoV is deployed on many GPUs distributed across many nodes. Aaby et al. investigated a latency hiding approach in ABMs for multinode, multi-GPU setups, but they experimented with only a single GPU per node [3]. Our work extends these earlier efforts by considering more a more complex supercomputer deployment and

a performance evaluation on larger jobs using multiple GPUs per node and with UPC++ for GPU-to-GPU communication.

A related approach to report the experience of optimizing ABMs is the work of Clascà et al. [11]. The authors describe optimizing PhysiCell, an ABM that simulates cells, substrates, and their environment using many computational threads. Their acceleration is performed on multiple CPU cores in a single compute node. Our work on SIMCoV-GPU builds on their work, extending ABMs to multi-node supercomputer environments and to multi-GPU clusters. Similarly, there is relevant work on ABM implementation for GPUs. FLAME GPU is a framework that allows the modeling of event based agent simulations on single GPUs [28]. Unlike these previous works, SIMCoV-GPU includes considerations for scaling on exascale supercomputers.

Other related work prepares other types of models besides ABMs for exascale computing. Such models include particle-in-cell simulations, chemical reaction simulations, thermodynamic flow simulations, and many others. Recent developments for HPC simulations in general include optimized octree construction for multi-GPU systems [22], adaptive-mesh refinement on fluid dynamics codes [13] and various physics simulations across diverse compute architectures [21, 32]. SIMCoV-GPU extends the field of HPC acceleration of scientific codes by addressing the unique challenges of ABMs that these other simulation frameworks generally do not face. In particular, SIMCoV-GPU reduces communication for autonomous agents on communication boundaries during the resource competition step.

6 DISCUSSION

Our performance evaluation shows that SIMCoV-GPU vastly outperforms SIMCoV-CPU, particularly on large problem sizes when the simulation has widespread viral/immune activity. In regards to scaling, we see some mixed results due to performance plateaus under certain conditions. In our strong scaling experiments we see evidence that the cost of parallelism on GPUs outweighs the performance gain of additional computational units when more than 16 GPUs are used. This can potentially be improved by optimizing the memory footprint of SIMCoV-GPU to allow for larger simulations to be executed on a small number of GPUs. We also see somewhat poor weak scaling until more than 16 GPUs are utilized. This suggests an initial cost of parallelism that requires sufficient problem size and computational units to overcome. This is not a significant issue for SIMCoV-GPU as it still out performs SIMCoV-CPU at all the problem sizes in the weak scaling experiment, and achieves a more healthy scaling behavior when many GPUs are used on even larger problems. On the other hand, our FOI scaling results are very promising. For a large simulation with many FOI causing high levels of activity, SIMCoV-GPU provides an 11.9x speedup over SIMCoV-CPU with a ratio of GPUs to CPUs of 32 to one. According to [2], the Perlmutter supercomputer has 32-bit floating point performance of approximately 75TFLOPS on GPU nodes and 5TFLOPS of CPU nodes, equating to a maximum speedup of 15.6x. Our weak scaling results show that as the size of the simulation space increases, SIMCoV-GPU achieves and maintains a four-fold advantage over SIMCoV-CPU. This result is encouraging for the the future use of SIMCoV-GPU to study viral infections throughout the lung. The total air volume of the average pair of healthy adult human lungs is approximately six liters [15], which is a rough estimate of the 3D space required for a true-scale simulation. With the default configuration of SIMCoV using five cubic micron voxels, this corresponds roughly to a simulation size of order 10^{13} voxels—far larger than any SIMCoV simulation run to date. To achieve this scale will require exascale supercomputers, and SIMCoV-GPU will enable us to use those resources efficiently. Once that scale of 3D space is achieved, other spatial topologies such as fractal branching airways can be easily tested by overlaying the topology on the voxels.

Our results on performance scaling with FOI show how SIMCoV-GPU will accelerate the scientific use of the model. An attractive use case and validation for SIMCoV is to use patient CT scans to initialize a simulation, then run the simulation and use it to predict disease trajectories—first to validate the model, and ultimately, perhaps for patient triage or to study possible interventions. CT scans of diseased patients do not contain point-like initial infection locations, but instead feature large patchy lesions that are distributed throughout the lung. Incorporating CT scans as initial conditions requires that many (hundreds, thousands, or more) SIMCoV voxels be initialized as FOI. In principle, with sufficient FOI, the simulation will saturate, in the sense that adding additional FOI won't degrade performance, because most voxels will already be active. Our results show that SIMCoV-GPU reaches this point much sooner than SIMCoV-CPU, demonstrating that SIMCoV-GPU reduces overhead and achieves high levels of parallelism. These results also set the stage for additional enhancements to the model, including new behaviors and parameters. For instance, more detailed modeling of the immune response, e.g., with additional immune cell and molecular components, or modeling airway dynamics, all of which would require additional computation per timestep.

SIMCoV is available as open source software and according to forks of the public repository, it is already being used as a platform for creating other ABMs. These ABMs include a simulation of large populations of ant-like foragers and a large-scale model of a coral reef ecosystem. SIMCoV-GPU will provide a straightforward path for these models to run on exascale supercomputers as well. Adapting new models that use SIMCoV as a platform to use SIMCoV-GPU instead will require developers to implement model behaviors as kernels. The diffusion, T cell movement, and T cell binding kernels cover a large range of possible behaviors that developers may need to implement from spreading concentrations to spatial competition for resources. The most likely behavior that would need a novel implementation is any kind of non-local interaction between agents. One idea for implementing this would be to define the spatial topology and the interaction topology as separate meshes which would allow the previous methods to work without much adjustment to algorithms or communication techniques. While our focus here was on large supercomputers, SIMCoV-GPU also benefits ABM development by allowing runs to be executed on personal computing GPUs on laptops or workstations. This allows researchers to treat their personal hardware as a virtual laboratory for developing smaller-scale ABMs that run efficiently. The SIMCoV-GPU project is available publicly and open source at http://bss.biodesign.asu.edu/projects/simcovgpu/.

6.1 Limitations and Future Work

Despite the near-optimal speedups we report in our evaluations, there is always the possibility of additional improvements, and in the future we plan to conduct additional performance evaluations across more nodes, which will likely reveal new opportunities for optimization. The performance evaluations we report in this paper are somewhat limited in the number of nodes tested, due to the computational resources we were able to access.

One interesting prospect would be moving to asynchronous agent updates, which would reduce the burden of communication within GPUs and across the GPU network. The T cell and epithelial cell behaviors in SIMCoV-GPU loosely resemble earlier cellular automata (CA) approaches to biological modeling which can sometimes be effectively computed asynchronously in parallel [23]. SIMCoV-GPU could also potentially benefit from dynamic domain decomposition, which would leverage interactions between CPU cores and GPUs. Large empty regions could then be quickly computed on the slowest hardware, using CPU processes for instance, while the available GPU workhorses rapidly compute the complex, activity-filled regions. Finally, regions of the simulation could be approximately computed when they are light on agent or environment activity. For instance, adaptive mesh refinement (AMR) could be applied such that regions without much activity can be computed at coarser scale, if the error cost was permissible [32].

7 CONCLUSION

This work presents SIMCoV-GPU, a simulation of viral lung infection that is deployed on multinode, multi-GPU supercomputers. Several GPU-focused algorithms and optimizations were required to achieve acceleration of the ABM, due to complexities arising from unpredictable load balancing, resource competition among agents, and the unique programming requirements of fast GPU codes. Our performance evaluation of the multinode, multi-GPU implementation reveals that as simulations become larger and more complex, the scaling performance of SIMCoV-GPU significantly dominates SIMCoV-CPU and achieves near-ideal speedups for the ratio of CPU-to-GPU that we studied. Beyond its relevance for SIMCoV itself, we hope that this work will pave the way for many other ABMs that can profit from running at exascale.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the partial support of NSF (CCF2211750, CICI 2115075, IOS-8044276), ARPA-H, and the Santa Fe Institute. The authors acknowledge Research Computing at Arizona State University for providing HPC and storage resources that have contributed to the research results reported within this paper. The authors acknowledge the support of the UPC++ developers group for their helpful advice and support. Authors from Lawrence Berkeley National Laboratory were supported by the Applied Mathematics and Computer Science Programs of the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] Frontier supercomputer debuts as world's fastest, breaking exascale barrier. https://www.ornl.gov/news/frontier-supercomputer-debuts-worlds-fastest-breaking-exascale-barrier. Accessed: 2023-10-23.

[2] Perlmutter architecture. https://docs.nersc.gov/systems/perlmutter/architecture/. Last Accessed January 25, 2024.

[3] Aaby, B. G., Perumalla, K. S., and Seal, S. K. Efficient simulation of agent-based models on multi-gpu and multi-core clusters. In *3rd International ICST Conference on Simulation Tools and Techniques* (2010).

[4] Andelfinger, P., and Uhrmacher, A. Optimistic parallel simulation of tightly coupled agents in continuous time. In *2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* (2021), IEEE, pp. 1–9.

[5] Axelrod, R. The dissemination of culture. *Journal of Conflict Resolution 41*, 2 (1997), 203–226.

[6] Axelrod, R., Daymude, J. J., and Forrest, S. Preventing extreme polarization of political attitudes. *Proceedings of the National Academy of Sciences 118*, 50 (2021), e2102139118.

[7] Axtell, R. L., and Farmer, J. D. Agent-based modeling in economics and finance: Past, present, and future. *Journal of Economic Literature* (2022).

[8] Bachan, J., Baden, S. B., Hofmeyr, S., Jacquelin, M., Kamil, A., Bonachea, D., Hargrove, P. H., and Ahmed, H. Upc++: A high-performance communication framework for asynchronous computation. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2019), IEEE, pp. 963–973.

[9] Borges, F., Gutierrez-Milla, A., Luque, E., and Suppi, R. Care hps: A high performance simulation tool for parallel and distributed agent-based modeling. *Future Generation Computer Systems 68* (2017), 59–73.

[10] Chumachenko, D., Dobriak, V., Mazorchuk, M., Meniailov, I., and Bazilevych, K. On agent-based approach to influenza and acute respiratory virus infection simulation. In *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)* (2018), IEEE, pp. 192–195.

[11] Clascà, M., Garcia-Gasulla, M., Montagud, A., Carbonell Caballero, J., and Valencia, A. Lessons learned from a performance analysis and optimization of a multiscale cellular simulation. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (2023), pp. 1–10.

[12] Collier, N., Ozik, J., and Macal, C. M. Large-scale agent-based modeling with repast hpc: A case study in parallelizing an agent-based model. In *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21* (2015), Springer, pp. 454–465.

[13] Davis, J. H., Shafner, J., Nichols, D., Grube, N., Martin, P., and Bhatele, A. Porting a computational fluid dynamics code with amr to large-scale gpu platforms. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2023), IEEE, pp. 602–612.

[14] DeAngelis, D. L., and Grimm, V. Individual-based models in ecology after four decades. *F1000prime reports 6* (2014).

[15] Delgado, B. J., and Bajaj, T. Physiology, lung capacity. *StatPearls [Internet]* (2022).

[16] Gerety, R., Spencer, S. L., Pienta, K. J., and Forrest, S. Modeling somatic evolution in tumoregenesis. *PLoS Computational Biology 2*, 8 e108 (2006).

[17] Harris, M., et al. Optimizing parallel reduction in cuda. *Nvidia developer technology 2*, 4 (2007), 70.

[18] Hernandez-Vargas, E. A., and Velasco-Hernandez, J. X. In-host mathematical modelling of covid-19 in humans. *Annual reviews in control 50* (2020), 448–456.

[19] Igehy, H., Eldridge, M., and Proudfoot, K. Prefetching in a texture cache architecture. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (1998), pp. 133–ff.

[20] Jennewein, D. M., Lee, J., Kurtz, C., Dizon, W., Shaeffer, I., Chapman, A., Chiquete, A., Burks, J., Carlson, A., Mason, N., et al. The sol supercomputer at arizona state university. In *Practice and Experience in Advanced Research Computing* 2023, pp. 296–301.

[21] Katz, M. P., Almgren, A., Sazo, M. B., Eiden, K., Gott, K., Harpole, A., Sexton, J. M., Willcox, D. E., Zhang, W., and Zingale, M. Preparing nuclear astrophysics for exascale. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), IEEE, pp. 1–12.

[22] Keller, S., Cavelan, A., Cabezon, R., Mayer, L., and Ciorba, F. Cornerstone: Octree construction algorithms for scalable particle simulations. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (2023), pp. 1–10.

[23] Li, J., Köster, T., and Giabbanelli, P. J. Design and evaluation of update schemes to optimize asynchronous cellular automata with random or cyclic orders. In *2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* (2021), IEEE, pp. 1–8.

[24] Lu, L., Nguyen, R., Rahman, M. M., and Winfree, J. Demand shocks and supply chain resilience: an agent based modelling approach and application to the potato supply chain. Tech. rep., National Bureau of Economic Research, 2021.

[25] Moses, M. E., Hofmeyr, S., Cannon, J. L., Andrews, A., Gridley, R., Hinga, M., Leyba, K., Pribisova, A., Surjadidjaja, V., Tasnim, H., et al. Spatially distributed infection increases viral load in a computational model of sars-cov-2 lung infection. *PLoS computational biology 17*, 12 (2021), e1009735.

[26] Ozik, J., Collier, N. T., Wozniak, J. M., Macal, C. M., and An, G. Extreme-scale dynamic exploration of a distributed agent-based model with the emews framework. *IEEE Transactions on Computational Social Systems 5*, 3 (2018), 884–895.

[27] Richmond, P. Resolving conflicts between multiple competing agents in parallel simulations. In *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part I 20* (2014), Springer, pp. 383–394.

[28] Richmond, P., Walker, D., Coakley, S., and Romano, D. High performance cellular level agent-based simulation with flame for the gpu. *Briefings in bioinformatics 11*, 3 (2010), 334–347.

[29] Sewall, J., Wilkie, D., and Lin, M. C. Interactive hybrid simulation of large-scale traffic. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (2011), pp. 1–12.

[30] Spies, T. A., White, E., Ager, A., Kline, J. D., Bolte, J. P., Platt, E. K., Olsen, K. A., Pabst, R. J., Barros, A. M., Bailey, J. D., et al. Using an agent-based model to examine forest management outcomes in a fire-prone landscape in oregon, usa. *Ecology and Society 22*, 1 (2017).

[31] Wang, S., Pan, Y., Wang, Q., Miao, H., Brown, A. N., and Rong, L. Modeling the viral dynamics of sars-cov-2 infection. *Mathematical biosciences 328* (2020), 108438.

[32] Zhang, W., Myers, A., Gott, K., Almgren, A., and Bell, J. Amrex: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications 35*, 6 (2021), 508–526.