2017 IEEE International Parallel and Distributed Processing Symposium Workshops

Exploratory Modeling and Simulation of the Evolutionary Dynamics of Single-stranded RNA Virus Populations

Jae-Seung Yeom Tanya Kostova-Vassilevska Peter D. Barnes, Jr. David R. Jefferson Tomas Oppelstrup
Lawrence Livermore National Laboratory, Livermore, California 94551 USA
{yeom2, vassilevska1, barnes26, drjefferson, oppelstrup2}@llnl.gov

***Abstract*—Infection, replication and mutation govern the population dynamics of viruses and are the key mechanisms driving their evolution. In particular, RNA viruses (such as the causative agents of Ebola, Dengue, Zika, West Nile, and SARS) have the highest mutation rates which enable them to form highly diverse populations within a single host, evade immune responses and develop resistances to drugs. Understanding the complexity of virus evolution is crucial for developing reliable countermeasures.**

We present an exploratory simulation model to study the evolution of heterogeneous virus populations in heterogeneous cell environments. This is a unique model that operates at three scales and captures the core mechanisms of the evolutionary process. To the best of our knowledge, this is the first HPC-based simulation of its kind.

I. Introduction

Mutation and selection are key mechanisms driving the evolution of viruses towards higher propensity to survive and reproduce in a given environment. In particular, RNA viruses have the highest mutation rates ($10^{-5} - 10^{-3}$ nucleotide misincorporations per genome replication), among all life forms. Such a high mutation rate results in high genetic diversity of a virus population even during a single host infection. The fast diversification of viruses leads to resistance to drugs and the low efficacy of vaccines such as annual flu vaccines, hindering the development of reliable countermeasures, such as for HIV, dengue, etc. It is, therefore, important to understand the complex dynamics of virus evolution.

The advent of next generation sequencing platforms opens up the possibility of understanding virus evolution at the molecular level. However, virus evolution is a multi-scale process, involving multiple entities, e.g. heterogeneous host cells and receptors, and complex intracellular and extracellular mechanisms. Understanding this complexity necessitates the development of a multi-scale model that is computationally challenging and thus requires high-performance computing (HPC).

Building a predictive model is extremely challenging. Many phenomena at the forefront of microbiology have yet to be explained. There are far more biological phenomena to be explored than currently understood. The challenges that microbiologists face also include the difficulty of experimenting with a particular aspect of a system in isolation, and the lack of tools to perform virtual experiments.

Thus, we take an exploratory modeling approach [1], and offer a novel computational tool so that the biologist can test what-if scenarios and gain insights into the evolutionary dynamics of RNA viruses in a cell culture with less effort and resources in a shorter amount of time. Our model algorithmically represents key known or hypothesized biological mechanisms in a computationally feasible way. The model does not aim to substitute for lab experiments, but to guide them and to speed up knowledge discovery.

We simultaneously simulate the extracellular and the intracellular activities of RNA viruses, explicitly represented as nucleotide (NT) and amino acid (AA) sequences, as well as the diffusion of virus particles among cells. The simulation produces the quasispecies [2] population evolved from given initial viral and cell populations as a result of genotype-specific replication. Given NT/AA frequencies such as those available in public databases [3, 4], our model tentatively scores each genotype's replication capabilities. This is a novel data-driven method, designed to explore hypotheses.

We adopt the optimistic parallel discrete event simulation paradigm [5]. We leverage the Rensselaer's Optimistic Simulation System (ROSS), which is based on the Time Warp synchronization mechanism [5–7]. We have implemented application-level reverse computation aided by a small amount of state savings as well as retraction of events that are no longer appropriate to execute.

We present two sets of demonstrations: i) the performance of weak scaling up to one million cells using 128K processors, and ii) the fitness composition of the quasispecies evolved from an initial Dengue virus population.

II. Background

A. Simulation approach

To capture the dynamic multi-scale phenomena in a parallel simulation, we rely on optimistic parallel discrete event simulations (PDES) [5]. Discrete event simulations are not time-stepped, i.e. they do not have constant simulation time delays between state changes, but instead dynamically calculate the simulation time delay between causally-related events (state changes) in the model. They tend to be dynamically irregular in time, often with a mixture of multiple time scales, and often stochastic.

978-0-7695-6149-3/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.139

There are two broad approaches to executing discrete event simulations in parallel, based on the style of synchronization used [5]. Conservative methods use conventional block-and-resume synchronization, but require good *lookahead* information to avoid deadlock and achieve a high degree of parallelism. Optimistic synchronization methods, in contrast, allow speculative computation and use *rollback* to correct causal errors [6]. They do not require any lookahead information, but are somewhat harder to implement, both in the simulator and in the model code. Rollback in an asynchronous distributed environment might seem complex or even impossible, but there is an elegant algorithm called *Time Warp* that performs it in a robust and general way [7].

B. Biological processes captured in the model

In this study, we focus on positive-sense single stranded RNA viruses, labeled as (+)ssRNA. The high-level description of the (+)ssRNA replication process is as follows. A virus particle (virion) may bind to a receptor available on a cell surface. If it does, it enters the cell and releases the positive sense RNA molecule which serves as a template for translation into proteins and copying to negative sense RNAs. The produced negative RNA is repeatedly copied back to a positive-sense genomic RNA, which in turn is packaged into a virion. With a certain probability, each new copy (negative or positive) may contain mutations. Cells have limited amounts of resources for replication, and thus can produce up to a fixed number of copies. When a maximum capacity is reached, the cell bursts and discharges the virions produced, which eventually diffuse to other cells.

Different genotypes likely exhibit different fitness among mutants, which is typically understood as the replicative capacity under a given environment [8]. This capacity depends on various capabilities, such as receptor binding or RNA copying by the polymerase. The efficacies of the mechanisms depend on the phenotypic properties of the participating proteins encoded by the viral genome.

III. Model Design

In this section, we describe the key components of our model and the events that update the model states representing important biological activities.

A. Model component overview

In our model, a population of host cells provides an environment for the evolution of a viral population. A viral load consists of virus populations of genotypes dynamically created by random point mutations as well as those predefined in the simulation input. We model the inter-cellular diffusion of the viral load as well as the simultaneous intracellular process of viral replication and mutation inside of each host cell. We denote a point location in a cellular population by a *pixel* (a cell site). Each pixel consists of zero or more cells and the extracellular space (or medium) representing the

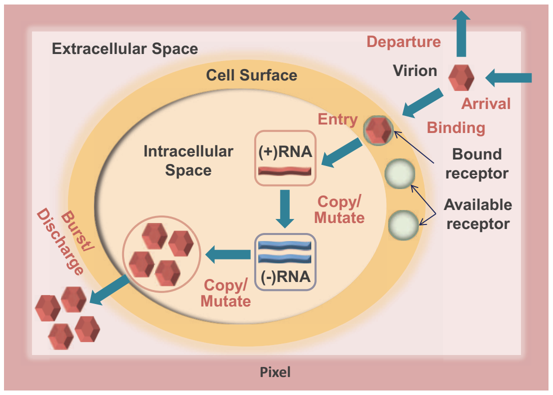


Figure 1. Overview of model components at a cell site (pixel) in a host cell population, and the model events.

well-mixed proximity space around the cells, as illustrated in Figure 1. The model supports heterogeneous types of receptors coupled with heterogeneous binding capabilities for distinctive genotypes. This enables the representation of the competition between genotypes in gaining access to the replication machinery of host cells. The model also supports heterogeneous cell types. Each cell type can have a different mix of receptors, in terms of the number and the type, as well as a different replication capacity. In addition, the distribution of cells across pixels can be non-uniform depending on the simulation input.

B. Compartment and virus tracking

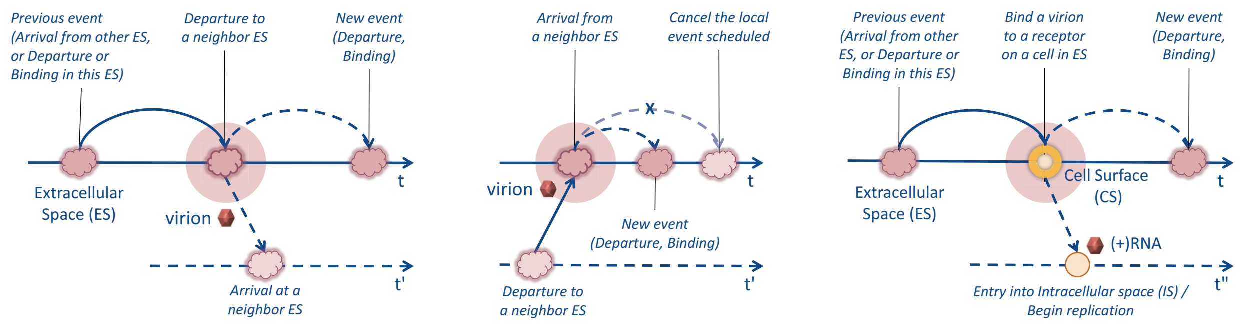
Instead of instantiating a per-virion object, we manage the particle count of each genotype per compartmental space. A pixel is divided into two types of compartments, the medium and the cell. A compartment enforces a capacity limit.

A cell is further divided into a surface, and an intracellular space. The former manages receptors available for binding. The latter contains a set of sub-compartments to track intracellular activities at various stages, such as the release of genetic material upon completion of entry, which initiates copying, and the production of mutated sequences after each replication stage.

The count-per-genotype tracking is useful in compressing the same kind of genotype instances into one data entry especially when the mutation rate is low, the sequence is highly conserved, or there is a large initial viral population. With highly-mutable and little-conserved sequences, this approach can require memory up to twice the amount used (requiring unnecessary count variables) by the alternative in which we instantiate individual virions.

C. Model event overview

The types of model events and the strategies to generate and handle them are discussed below. In a medium, we schedule the earliest virion event for the whole viral load


Previous event (Arrival from other ES, or Departure or Binding in this ES)
Departure to a neighbor ES
New event (Departure, Binding)
Extracellular Space (ES)
virion
t
Arrival at a neighbor ES
t'
Arrival from a neighbor ES
Cancel the local event scheduled
x
virion
t
New event (Departure, Binding)
Departure to a neighbor ES
t'
Previous event (Arrival from other ES, or Departure or Binding in this ES)
Bind a virion to a receptor on a cell in ES
New event (Departure, Binding)
Extracellular Space (ES)
Cell Surface (CS)
t
(+)RNA
Entry into Intracellular space (IS) / Begin replication
t''

i) decrease the number of departing virus
ii) schedule an arrival event at the destination
iii) schedule a new local event

(a) A departure event (local)

i) increase the number of arriving virus
ii) retract the outstanding local event
iii) schedule a new local event

(b) An arrival event (non-local)

i) decrease the number of binding virus
ii) decrease the number of binding receptor
iii) schedule an entry event in the cell
iv) schedule a new local event

(c) A binding event (local)

Figure 2. Sketches of extracellular event handling. A circular shade indicates the current event under processing in the timeline. A dashed curve leads to an event scheduled by the current event. A solid curve shows which past event scheduled the current event.

of the medium one at a time, instead of independently scheduling for each virion. While this is primarily for efficiency, it is also natural to manage them in a collective fashion for modeling binding competition.

In general, our strategy in scheduling an extracellular event is to determine only the necessary details for scheduling, such as the time of the next event, and determine the details as late as possible. Then, we determine the rest when we execute it; e.g. which virus binds to which receptor. This lazy specification of event is useful in two ways. First, it helps to avoid wasting computation by rollbacks in optimistic PDES. Second, this allows us to define an event based on the most up-to-date model state.

A *local event* denotes an event scheduled locally to update the local state, such as a virion departure and a virion-receptor binding as shown in Figure 2. An arrival is an example of a non-local event. We determine the time of the next local event by estimating the interval with no state change in the local system. As there are multiple types of local events with unique event rates, we make independent estimation for each (Section III-D and III-E), and then choose the earliest one for scheduling. Note that we have no estimation for non-local events. An externally scheduled event may arrive during such a period and update the local state as shown in Figure 2 (b). This invalidates the earlier local estimation. In this case, we *retract* the local event scheduled based on the invalidated estimation (Section IV-B). After the state update, we schedule a new local event based on the new state. We maintain one outstanding local event unless there is no local virion. Without one, we schedule no local event. This is analogous to defining the default behavior given that nothing unexpected occurs.

In a cell, we individually handle the entry, which is the beginning of a copying stage, as well as the successive copying stages for each RNA instance (Section III-F). Whether a copying results in mutation or not is determined upon completion of the copying. Finally, we schedule a burst event when the intracellular viral load exceeds the predefined limit of the cell, which in turn discharges the progeny virions into the surrounding medium.

D. Diffusion

A virus particle may move to a random neighbor pixel at a random point in time. While we explicitly model the diffusion of a virion, the random decision is made per pixel rather than per virion for computational efficiency. The diffusion occurs across pixels, not within a pixel even if there are multiple cells in a pixel. In scheduling, we determine the time of a next departure from the pixel. During execution, we decide which virion should diffuse, and its destination pixel. We further explain how we determine the diffusion variables here.

We represent a pixel as a vertex in a graph, with edges connecting neighboring pixels. Each edge is directional and parameterized by the mean time to reach the other end ($\overline{\Delta t_r}$) and the *diffusion propensity factor* (f^D). We randomly choose a neighbor for a virion to move to, considering the propensity factor which represents the non-homogeneous degree of diffusion to the neighbor [9]. The higher the value, the greater the chance of being selected. The time interval to the next diffusion in a medium, Δt_d, depends on the current viral load in the pixel and the diffusion propensity factors for neighbor pixels. Such an interval is determined randomly from the exponential distribution with a mean $1/\lambda_d$ as $\lambda_d exp(-\lambda_d t)$ where

$$\lambda_d = \sum_{v\in \mathbb{v}} \sum_{m\in \mathbb{m}} f^D_{m,v} \times cnt^V_v ,$$

v is a virus type in $\mathbb{v}$ which is the current set of virus types in the medium, m is a neighbor in $\mathbb{m}$ which is the current set of neighbors of the medium, cnt_v^V is the number of virions of type v, and $f_{m,v}^D$ is the diffusion propensity factor for v to m. Currently, $f_{m,v}^D$ is an input parameter which needs to be set for a particular biological experiment. However, such a parameter can be dynamically updated to reflect environmental changes using events. Once we pick a random interval Δt_d, we can use it to schedule a departure event for $t = t_{cur} + \Delta t_d$ where t_{cur} is the current time. However, whether to actually schedule or not depends on the estimated time of the other local event, binding, in the medium as explained in Section III-C.

We randomly determine the virion to depart and its destination based on the current state of the pixel when we executes a departure event as opposed to when it is scheduled, as explained in Section III-C. To make the choices, we rely on a similar method described in Section III-E. Finally, we schedule an event of arrival at the destination for $t = t_{cur}+\Delta t_r$, where Δt_r is a randomly determined interval from the distribution of time-to-reach, $\lambda_r exp(-\lambda_r t)$ where $\lambda_r = 1/\overline{\Delta t_r}$. As a virion leaves a pixel and moves into a new pixel, we decrease the count of the virus type at the source medium and increase it at the destination. Figure 2 (a) and (b) illustrate the process as well as the generation/retraction of a local event discussed in Section III-C.

E. Receptor binding

Our binding model is simplified in that a virion directly binds to a receptor on a cell instead of attaching to the surface of a cell in advance. While an elaborate model can be incorporated in the future [10], we concentrate on the effect of non-uniform binding capability of genotypes on the population dynamics. We define the *binding fitness* $f_{r,v}^B$ of a genotype v as a relative measure of binding capability to a particular receptor of type r in a pixel. We use this parameter to compute when a virion will bind, and to decide the entities involved in the event at execution; i.e., which virion to bind to which receptor of which cell in the pixel.

We compute binding weights in multiple forms to aid various random decisions. Suppose that we have a set of genotypes $\mathbb{v}$, a set of receptor types $\mathbb{r}$ and a set of cells $\mathbb{c}$ in a pixel. We define the weight of binding genotype v to a single receptor of type r independently of the cell as

$$W_{r,v} = f_{r,v}^B \times cnt_v^V \ ,$$

where cnt_v^V is the number of virions of type v. We also define the weight of binding to a single receptor of type r independently of the genotype as $W_r = \sum_{v\in\mathbb{v}} W_{r,v}$. Then, the weight of binding to any receptor of type r on cell c is

$$W_{c,r} = W_r \times cnt_{c,r}^R \ .$$

where $cnt_{c,r}^R$ is the number of receptors of type r on cell c. Similarly, the weight of binding to a cell c is

$$W_c = \sum_{r\in\mathbb{r}} W_{c,r} \ .$$

Then, the total weight is $W_{tot} = \sum_{c\in\mathbb{c}} W_c$.

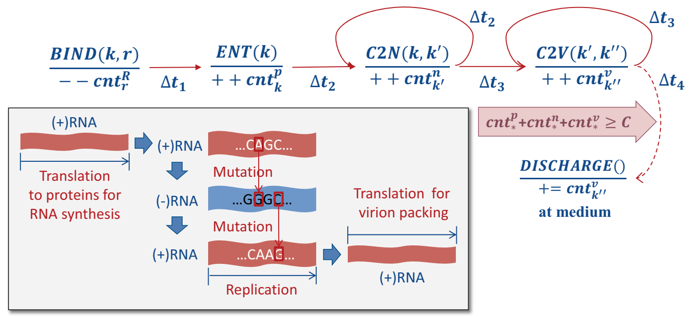
Scheduling a binding event is similar to scheduling a departure discussed in Section III-D. Once we obtain Δt_b from the random distribution $\lambda_b exp(-\lambda_b t)$, where $\lambda_b = W_{tot}$, we can schedule a binding event in the medium. However, whether to actually schedule depends on the estimated time of the other local event, a departure (see Section III-C).

Upon binding, we schedule the completion of entry in the cell for $t = t_{cur}+\Delta t_e$, where Δt_e is randomly chosen from the distribution of time-to-enter; i.e. $\lambda_e exp(-\lambda_e t)$ where $\lambda_e = 1/\overline{\Delta t_e}$. Section III-H describes how $\overline{\Delta t_e}$ is obtained per genotype. As a virion binds to a receptor on a cell, we decrease the count of the genotype at the medium as well as the number of available receptors of the chosen type on the chosen cell. Figure 2 (c) shows the process. In Section IV-A, Figure 5 shows how we keep binding weights up-to-date.

F. RNA copying

As discussed in Section II, copying a positive-sense ssRNA involves two stages: copying from a (+)sense RNA to a (-)sense RNA followed by copying from a (-)sense RNA to a (+)sense RNA as depicted in Figure 3. The latter further entails a packaging stage which turns the (+)sense RNA into a progeny virion. We label the former as $C2N$, and the latter as $C2V$. The first stage begins with a translation phase enabling RNA synthesis followed by a replication phase. The second stage replication is followed by a translation to enable virion packing. In our current model, we simply add the duration of both phases to compute the time interval to complete a copy stage as $\overline{\Delta t_c}$=$\overline{\Delta t_{tr}}$+$\overline{\Delta t_{rp}}$ where $\overline{\Delta t_{tr}}$ and $\overline{\Delta t_{rp}}$ are times taking to translate and replicate respectively. These are genotype-dependent and unique to the stage. Section III-H describes how these parameters are obtained. We schedule a copy event, such as $C2N$ or $C2V$ which represents the completion of a copying stage, using the respective $\overline{\Delta t_c}$. We currently rely on a random distribution $\lambda_c exp(-\lambda_c t)$ to obtain Δt_c, and compute the event time as $t = t_{cur} + \Delta t_c$, where $\lambda_c = 1/\overline{\Delta t_c}$. Experimental data is necessary to set up a more appropriate distribution.

Figure 3 shows the chain of intracellular events, i.e., entry, copy, and burst events. The binding ($BIND$) event schedules an entry (ENT) event, which is the beginning of intracellular chain leading to $C2N$ and $C2V$. The first step in executing a copy event is to determine if the RNA copy is a mutant as we discuss in Section III-G. If so, we register it into a local database (see Section IV-D). We increase the count of the new genotype in the associated compartment. Then, we schedule subsequent copying events. In case of $C2N$, two following events are scheduled. One is for another round of copying the original (+)sense RNA. The other is to copy the



Copy event handling ($C2N$ and $C2V$)
i) Determine if the RNA copy is a mutant. If so, compute mutation and register it to local database.
ii) Increase the count of genotype at the associated compartment. If internal load reaches a limit, schedule a burst event, and skip iii) and iv).
iii) For $C2N$, schedule another $C2N$.
iv) For $C2N$ and $C2V$: schedule $C2V$.

Figure 3. The chain of intracellular events in the model, such as an entry and the two staged copying; copying from a (+)sense RNA to a (-)sense RNA, and copying from a (-)sense RNA to a (+)sense RNA. Each copying stage repeats until the cell bursts as the internal load reaches the threshold.

resultant (-)sense RNA of the current stage. In case of $C2V$, only the initial (-)sense RNA of the current stage is subject to further copying. In case that the total internal population or the number of progeny virions reaches a threshold provided by a user, a burst event is scheduled, which coincides with the discharge in the medium.

G. RNA mutation

We choose a random number r of mutated positions during a copy from the cumulative binomial distribution as:

$$\sum_{k=0}^{r} \binom{n}{k} q^k (1-q)^{L-k}$$

where L is the length of the sequence, and q is a mutation rate. If $r > 0$, we choose r uniformly random positions. Then, we decide the new NT using the transition probability p_{tr}, which biases the changes between particular NT pairs. Both q and p_{tr} are simulation inputs.

Currently, we only model point mutation, and plan to add other mutation mechanisms. Recombination is known to be important for some virus species and mix of species while being less common for Dengue virus in a single host [11].

H. Fitness computation

To quantitatively describe how a genotype is capable of surviving in a given environment, we require users to provide score functions for binding, entry, and copying. Then, we scale the parameters of a reference genotype by the relative scores for each mutant. Users provide the parameters of the reference sequence as input. For parameters unique to the environment, we require them for each environment (e.g. receptor or cell). Here, for demonstration, we tentatively compute the relative fitness scores considering the mutations at the NT or AA positions affecting these functionalities, especially on those corrupting conserved subsequences, as well as the NT/AA frequencies reported in public databases [3, 4]. This data-driven method enables us to explore hypotheses and gain insights. We plan to investigate if we can refine scoring methods by combining further bioinformatics knowledge [12].

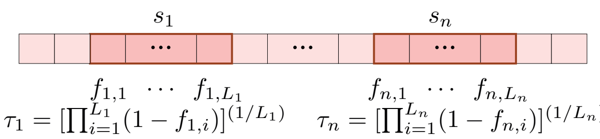

s_1
s_n
$f_{1,1}$ ⋯ f_{1,L_1}
$f_{n,1}$ ⋯ f_{n,L_n}
$\tau_1 = [\prod_{i=1}^{L_1}(1-f_{1,i})]^{(1/L_1)}$
$\tau_n = [\prod_{i=1}^{L_n}(1-f_{n,i})]^{(1/L_n)}$

Figure 4. Per-mutation tolerance τ_k pre-computed based on the observation frequency $f_{k,i}$ of a conserved subsequence s_k of length L_k.

We express the parameter of relative binding fitness $f_{r,v}^B$ as a rate, and the rest of the parameters as mean-time-to-event (MTTE). A binding fitness $f_{r,v}^B$ is necessary for processing an arrival event as we update binding weights as shown in Section III-E. Handling a binding event requires the MTTE of entry ($\overline{\Delta t_e}$) to schedule an entry. The parameters for copying consist of four MTTE values, corresponding to the translation ($\overline{\Delta t_{tr}}$) and the replication ($\overline{\Delta t_{rp}}$) in $C2N$ and $C2V$ respectively.

How we compute a relative score is described below. We gather data on conserved subsequences from public sources. Especially, we obtain the NT frequencies per base position and the AA frequencies per codon position using the software introduced in [4]. Some studies identify conserved AA subsequences in coding regions and present the observed frequencies of those [3]. Others identify the conserved NT subsequences especially in the non-coding regions and in the coding regions adjacent to them [13–18]. We combine these data to acquire the frequency of each known conserved subsequence. We also define a conserved AA as the AA of a set of synonymous codons whose sum of observed frequencies exceeds a threshold, which is set to 80% in this demonstration to be consistent with the data in [3].

Figure 4 shows the main idea of our data-driven strategy to decide relative fitnesses. For a mutation that falls onto a conserved subsequence s_k, we define the *per-mutation tolerance* of s_k as $\tau_k = g_k^{(1/L_k)}$, where L_k is the length of s_k, and $g_k = \prod_{i=1}^{L_k}(1 - f_{k,i})$ while $f_{k,i}$ is the observed frequency of the (collectively) conserved value at position i in s_k. A position represents either an AA or an NT. A subsequence of length 1 has a single position. A mutant sequence may have multiple mutations that possibly belong to different subsequences. We define the *per-mutant tolerance* tol as the product of all the per-mutation tolerance associated with the mutated positions. Given M mutations,

$$tol = \prod_{i=1}^{M} \tau_{\sigma(i)}$$

where $\sigma(i)$ is the index of the conserved subsequence to which the i_{th} mutation belongs. If a mutation does not belong to a conserved subsequence, $\tau_{\sigma(i)}$ is 1.

Finally, we compute the scaling factor for a rate parameter as $tol/(1-tol)$, and that for a time parameter as $(1-tol)/tol$. The conserved subsequences/positions are categorized by the regions affecting a particular function of the virus. For example, in Dengue virus parts of the C and E coding regions affect binding, the prM region affects entry, and non-structural coding regions and non-coding regions affect copying [13–18]. The tolerance is independently computed for each functionality. Thus, a region can contribute to multiple functions. If tol is zero, the sequence has no capability of the function(s). In a purely random position, f_k is 0.25 as there are four types of NTs: A, C, G, and U. Additionally, a mutation changing an ordinary codon to a stop codon or vice versa leads to a non-functional sequence. We also provide a capability to handle a group of conserved subsequences/positions in case that there exist multiple subsequences that are effective over the same positions. In this case, we simply treat the sum of the frequencies of a group as a whole.

IV. Parallel Implementation

In this section, we discuss how we map our model to ROSS [7], and the challenges overcome in the effort.

A. Rollback

Rollback is a mechanism to correct a causal error from speculative execution in optimistic PDES. Modern optimistic simulators, such as the ROSS, implement rollback partly by requiring that the model be written using reversible code. Most ordinary code is irreversible, meaning that it destroys information as it executes. Information is often lost when data is overwritten, for example, or when memory is freed and its contents are lost. As a result, it is not possible to restore a prior state using just the information in the current state. However, reversible code uses a stack to save information which would otherwise be destroyed during speculative execution. When rollback to a prior state is required, saved data can be popped off the stack and restored to reconstruct a prior state as needed.

We implement our model code to be reversible, saving the least amount of information necessary to reconstruct the prior state. We leverage the ROSS event message itself, which is kept by ROSS on a rollback stack, to save our minimal prior state information, and manually implement reverse

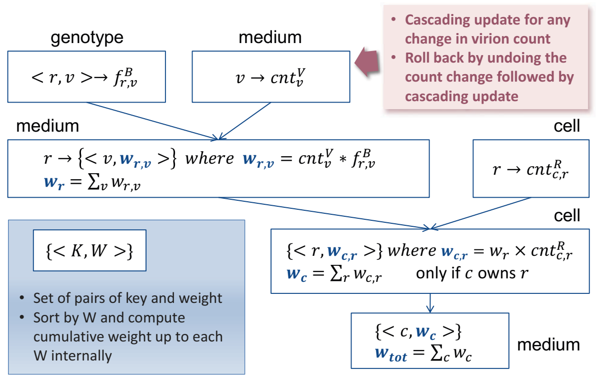

genotype
medium
$< r, v > \rightarrow f^B_{r,v}$
$v \rightarrow cnt^V_v$
Cascading update for any change in virion count
Roll back by undoing the count change followed by cascading update
medium
cell
$r \rightarrow \{< v, w_{r,v} >\}$ where $w_{r,v} = cnt^V_v * f^B_{r,v}$
$w_r = \sum_v w_{r,v}$
$r \rightarrow cnt^R_{c,r}$
cell
$\{< K, W >\}$
Set of pairs of key and weight
Sort by W and compute cumulative weight up to each W internally
$\{< r, w_{c,r} >\}$ where $w_{c,r} = w_r \times cnt^R_{c,r}$
$w_c = \sum_r w_{c,r}$ only if c owns r
$\{< c, w_c >\}$
$w_{tot} = \sum_c w_c$
medium

Figure 5. Management of binding weights per pixel discussed in Section III-E. The medium keeps track of cnt^V_v and each cell maintains $cnt^R_{c,r}$. The medium computes $W_{r,v}$ and W_r independent of cell, and updates them when there is a change in cnt^V_v. This typically occurs with an arrival, a departure, a binding, and a discharge event. A cell computes $W_{c,r}$ and W_c, and updates them when $cnt^R_{c,r}$ or W_r changes only if it owns a type r receptor. The medium updates W_{tot} with any change in W_c.

computation routines to restore the complete prior state. The event message will be thrown away as the global virtual time (GVT) progresses past the time of the event. At that point we will no longer require a rollback to the state preserved in the message. Figure 5 shows an example. We only save a single count value to restore the previous state of entire weight lists. To rollback the cascading update, we simply undo the count change which triggered the update and then recompute the weights. This also ensures reproducibility by avoiding the side effects from the otherwise reversed order of floating point operations in reverse computation for rollback. For reversing a mutation, we remove the mutant info added to the local database as well as other associated data.

B. Retraction

Almost all events schedule one or more other events to be executed at a future simulation time. If an event e_1 at time t_1 schedules another event e_2 for a time $t_2 > t_1$, then e_2 will normally execute when the simulation reaches time t_2. However, sometimes at an intermediate time t_3 ($t_1 < t_3 < t_2$) conditions may have changed so that it is no longer appropriate to plan to execute event e_2 after all. Then the scheduled event e_2 can be retracted, and the simulation will proceed as if it was never scheduled in the first place. This is different from rollback in that the event has not occurred yet. When retraction is used, all scheduled events must be regarded as tentatively planned. They execute only if they are not retracted until the planned execution. For example, a model may specify a default behavior at some point in the future unless a certain event occurs before then.

ROSS does not support retraction directly. However, we program the same effect as follows. We keep track of the outstanding local event (Section III-C) by the pointer to the event which becomes a part of model state. Fortunately, there is at most one local event at a time in our model that is subject to retraction. To retract it, we simply mark it as 'retracted'. We write the model event handler to ignore such an event. If the other event which caused retraction is rolled back, we undo the retraction marking on the retracted event, which we can access via the pointer we store.

C. Processor mapping

A logical process (LP) [7] simulates a pixel. We assign one kernel process (KP) per MPI rank [7]. Each KP manages an event queue for one or multiple local LPs. Also, local LPs share the address space, we do not share model data among different LPs except for the initial simulation input and for the post-processing of simulation results. Currently, all the model compartments are on the pixel LP. However, in the future, we plan to separate the cell and the medium into different LPs on different KPs. In this scenario, we would still keep the cell surface as a part of the medium such that binding events are isolated in the medium LP while the intracellular events experience a minimum disturbance from the extra-cellular activities.

D. Sequence data management

We store a mutant sequence in a compressed format. An *edit list* contains only the positions that differs from the reference. Each entry in a list is a pair of a mutated position and the new NT at the position. This is packed into 4-bytes reducing the amount of memory needed. We also compress a full reference sequence by packing three NTs into a single byte. The initial reference sequence of each species and its id are globally known.

The diffusion of a locally mutated sequence requires sufficient pieces of information to be encoded in a message such that a receiver can reconstruct the sequence. A basic sequence descriptor in a message consists of the edit list and the id of the reference sequence. For further optimization, we maintain a pair of cache structures. One is maintained by a receiver to store the pairs of the source of a message and the sender's local id of the sequence received. The other is maintained by a sender to store the list of receivers of each sequence it has transmitted. We label the former as Map_r and the latter as Map_s. We substitute it with the pair of the local sequence id at the sender and the id at the receiver side if known.

We make use of the cache structure as follows. First time a sender sends a sequence, in principle, has to send the local id, the edit list and the full reference sequence. For subsequent communications, the sender only sends the local id. If a sender knows that a receiver already has the reference, it sends the local id, the edit list and the reference id. A sender knows if it is the case if the sequence originally came from the receiver (according to Map_r), or it has previously sent it to the receiver and, most importantly, the receiver has received it (according to Map_s). If the time of virus arrival has not reached at the destination, the receiver has not seen it. Thus, we update Map_s via a local event scheduled at the same time of arrival. A sender attempts to translate the local id to the counterpart at the destination whenever possible by using Map_r. Coordinating this distributed data structures as well as the sequence database is especially challenging as it needs to be reversible.

We implement the sequence database based on `boost::multi_index` to allow identification of a sequence by various search criteria including the full sequence, the edit list, and the id while taking advantage of early termination of comparison by using light-weighted xor-based hashing.

E. Scalable data reduction

At the end of the simulation, a viral population with high genetic diversity remains on each MPI rank. Moreover, the same genotype appears on multiple ranks under different local ids. To reduce the data and collect statistics of each genotype in an efficient and scalable manner, we rely on an approach combining a distributed hash table with communication scheme based on a balanced tree.

In post-processing, each gene sequence (genotype) is identified with its unique fingerprint, which is fixed in size and far smaller than the full sequence. A strong hash function (160 bit SHA-1) generates a fingerprint such that the probability of non-unique fingerprint is vanishingly small.

We aggregate statistics of each genotype by a distributed hash table including information such as specimen counts, fitnesses, and mutation counts. We denote such information combined with the genotype fingerprint a record. The destination MPI rank of a record is defined as the modulo of the fingerprint by the number of MPI ranks.

With destinations identified, we distribute the records in a scalable way such that no MPI rank sends or receives too many record or runs out of memory. We accomplish this by a recursive bisection strategy, in which each process sends and receives $O(\log_2 N)$ messages to the counterpart in the other section. This requires memory enough to store $O(S \times \max_i (N_i^{\text{before}}, N_i^{\text{after}}))$ records, where S is the size of a record, N is the total number of record, $N_i^{\text{before}}/N_i^{\text{after}}$ is the number of records owned by process i before/after data distribution. The hash function helps to ensures that the resultant data distribution is relatively uniform. To make sure no process runs out of memory, we balance data per section as needed before each recursion step.

As a result of exchanging data, all records pertaining to the same genotype are gathered on the same processor, enabling us to reduce records per genotype. For each aggregated record, we choose an arbitrary destination among the processes that contributed statistics for this genotype. We return the aggregated records to the chosen destinations by the same recursive data distribution. This ensures that the destination process has a complete copy of the genetic information of this genotype. Finally, we rely on MPI-IO to write aggregated population statistics and complete genetic information for each genotype that existed in the simulation.

V. Results

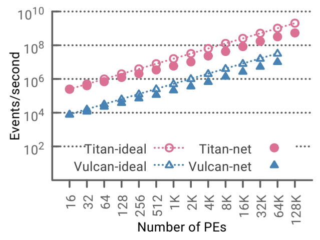
We conducted weak scaling evaluations on two HPC platforms, a Cray XK7 cluster (called Titan) [19], an IBM BlueGene/Q cluster (Vulcan) [20]. These are the state-of-the-art HPC systems, and respectively ranked as the 3rd and the 21st fastest system in the current (Nov. 2016) Top500 list [21]. Figure 6 shows the results. For all these runs, we use 16 MPI tasks per compute node. We execute the model with two cells per pixel LP, and four LPs per MPI task. Thus, total eight cells are simulated per processing element (PE) used. On the Titan cluster, we run up to 128K processors simulating total one million cells. Each simulation runs until all the cells burst, at which point there exists no cell for a virion to infect. Until then, a virion released from a burst cell may infect a cell that is alive and has a receptor available or may continue to diffuse to another cell site. In this study, we simulate Dengue serotype 2, which has an approximately 11K-base-long sequence. The parameters of the virus are mostly obtained from [22, 23]. We define two different types of cells with different mix of receptors for the demonstration. We use 0.5 for the multiplicity of infection (MOI) parameter, meaning that there is one initial virus particle per two cells. The larger the MOI, the faster the simulation runs as cells get infected quicker and thus burst earlier. For example, compared to the problem with MOI=1, it runs 22 times faster with MOI=1000 and 3.6 times slower with MOI=0.001 by the simulation clock. For Titan, we use 8MB hugepages setup. For final file output, we use non-blocking MPI-IO mode with individual file pointers on Lustre storage systems.

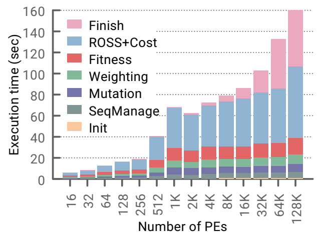
Figure 6 (a) shows net event rates. A net rate is computed as $R_{net} = (E_{tot} - E_{rt} - E_{rb})/T$, where E_{tot} is the total number of events, E_{rt} is the number of retractions, E_{rb} is the number of rollbacks, T is the time to execute excluding the initialization and the post processing. The highest net event rate achieved on Titan is about 540 *million events per second* (meps) using 128K PEs. On Vulcan, 11 meps is achieved using 64K PEs. The gross rate, computed as $R_{gross} = (E_{tot} - E_{rb})/T$, is higher than the net rate by 13% on average for both platforms. The ideal event rate shows the linearly increased rate as more PEs are used. It is computed as $R_{ideal} = R_{net}(16) \times N_{PE}/16$, where $R_{net}(16)$ is the net event rate using 16 PEs on a single compute node, and N_{PE} is the number of PEs used.

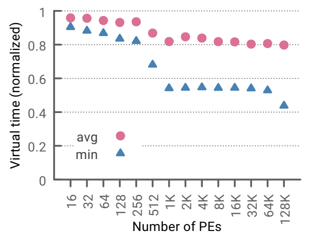
Figure 6 (b) shows the execution times from the weak scaling test on Titan. The total execution time including IO increases by a factor of 33 as the problem size becomes 8192 times larger on Titan, and by 21 times as it becomes 4096 times larger on Vulcan. The breakdown shows the amount of time spent by each part of the code. Before and after the event processing loop of a simulation are initialization and post processing. *Init* shows the maximum time taken across PEs for the initialization and data loading. *Finish* shows that for post-processing, i.e., the reduction of the sequence information as described in Section IV-E, as well as result file writing. The largest file written is about 4.3 GB. We report the averages times spent across PEs for those related to asynchronous event processing as follows. *Fitness* shows the time spent for fitness computation discussed in Section III-H, and *Mutation* shows that for mutation discussed in Section III-G. *SeqManage* shows the time taken to manage/search sequence information discussed in Section IV-D. *Weighting* shows that for binding weights and the diffusion weights discussed in Section III-D and III-E. Finally, *ROSS+Cost* shows the rest of total time, which consists of the parallelization cost including event handling, message packing/unpacking and GVT synchronization as well as the effect of load imbalance.

The cost *ROSS+Cost* is highly dependent on the model and the load balance, especially on their stochastic nature. Figure 6 (c) shows the average and the minimum of the virtual time of cell burst events, which is normalized by the virtual time of the last burst. As the problem size increases up to 8K cells (1K PE), so is the difference between the virtual times of the earliest and the last cell bursts. This increases the inherent chance of rollbacks as discharged virions diffuse to other pixels which may have progressed to different virtual times. Imbalanced load between processors as well as message delay may lead to rollbacks as the time of an event generated at one processor (possibly with a higher load) is already past where the event supposed to occur. We observe that the ratio of rollbacks increases from 60% to 80% of the total events. Beyond 8K cells, the difference remains steady until 1M cells. Figure 6 (d) shows the maximum, the average, and the minimum of the wallclock times of cell bursts. The wallclock time of the last burst increase as the number of processors increases even when the range of the virtual times of cell bursts is steady. The correction of a causal error further leads to an increased execution time, and complicates load analysis. We need to gain a better understanding on how to define a load in the presence of rollbacks and retractions. The new version of ROSS [24] is being built on top of Charm++ to enable dynamic load balancing for improving performance scaling.

Figure 7 shows the impact of two ROSS runtime parameters on performance: the batch processing size and the GVT synchronization interval. The former defines the upper bound on the number the events in the queue to process before incrementing the synchronization trigger count. When the count reaches the latter value, the runtime relies on global reductions to determine the GVT progress. We experimentally determine that it is optimal when both parameters are set to 128 on single node runs. For memory limitation, we


(a) Net event rates


(b) Execution times (Titan)


(c) Virtual times of cell bursts (Titan)

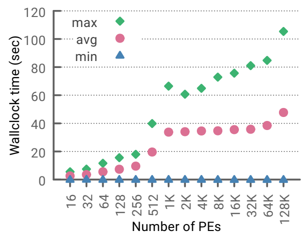

(d) Walltimes of cell bursts (Titan)

Figure 6. Weak scaling performance using MOI=0.5 and 8 cells/PE.

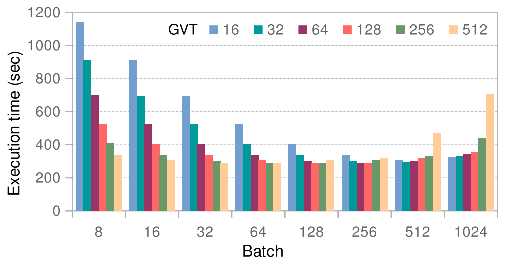
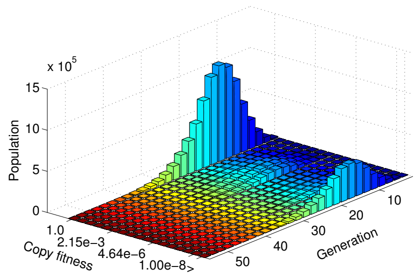


Figure 7. Impact of ROSS parameter choices on performance: GVT synchronization interval vs. batch. Tested using 10K cells, 128 LPs/rank, and 40 MPI ranks on a 40-core node of Intel E7-4870 with 2TB memory.


(a)

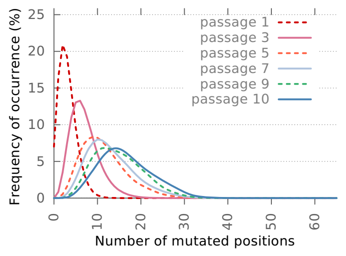

(b)

Figure 8. A passage experiment. (a) At the end of 10th passage with MOI=1, 54% of the population has the intact copy fitness score ($\frac{tol}{1-tol}$), most of which contain mutations on non-conserved NT or AA, or that lead to synonymous codons. (b) The number of point mutations vs. the number of passages.

set the batch size to 128 and the other to 64. While these are reasonably good, the optimal choice depends on the model, which is stochastic in nature, and the execution platform.

Figure 8 demonstrates a serial passage experiment, in which a viral population sample obtained from the previous infection stage (passage) is used to infect a new cell culture in the next stage. Figure 8 (a) shows roughly 46% of the entire population consists of mutants that have decreased copying fitness score $\frac{tol}{1-tol}$ as a result of mutating conserved regions and negatively affecting the functionality. Figure 8 (b) shows how the viral population evolves through 10 passages in terms of the number of mutations.

VI. Related Works

To our best knowledge, no comparable HPC simulation model of virus evolution exists in the literature. In general, multi-scale models are fairly rare or small scale.

A majority of the published simulations are of simple small-scale stochastic models. For example, Lazaro et al. [25] present a time-stepping model with no explicit sequence structure; the model is used to study the evolution of fitness in the context of quasispecies theory. Similarly, Jenkins et al. [26] consider a model simulating pseudo genomes of 100 bases that replicate with mutation. The reference genome and its neutral mutants (neutral sites are predefined) have fitness 1, while the rest have fitness 0.1. No cells are included in the model. Ribeiro et al. [27] complement a simulation with clinical observations to study the diversification of Hepatitis C virus. However, all mutations are neutral in terms of fitness. Nonacs and Kapheim [28] include more biological mechanisms and simulate genomes with 9 loci existing in 10 different alleles. Cells can be infected by 1-7 randomly chosen viruses; subsequent rounds of replication; at each round 120 cells get infected and release virus simultaneously; there is intracellular competition between genotypes, but no explicit fitness is associated with different genotypes. Included is a simulated immune response.

The closest in spirit HPC simulation model that we have found in literature [29] simulates the evolution of bacterial communities. The model simulates the molecular pathways within individual cells and the effect of mutations on gene expression and changing the bacterial phenotype. Xia et al [30], offer an HPC model to study the effect of mutations on the structure of the influenza virus. This model first deduces influenza evolutionary changes from yearly data and then performs molecular dynamics simulations to attempt to predict the affinity of the mutated virus to the human cell receptor. Da Silva and Hughes [31] simulates the dynamics between hypothetical antibody reaction on mutations of 96-base-long HIV-1 V3 region without involving cells. Woo and Reifman [32] present a model that simulates evolutionary dynamics with several amino acids using empirically inferred fitness landscape.

VII. Conclusion and Vision for the future

We provide an *in silico* tool to allow faster and inexpensive exploration of the evolutionary dynamics of RNA viruses. It is designed to provide insights that could guide lab experiments to test hypotheses, and to speed up knowledge discoveries. The plans for future work include incorporating immune responses into the model, enabling additional mutation mechanisms, scaling up to the organ level, and modeling cross-species transmissions. The model can serve as a foundation for studying various drug interventions, defective interfering particles, and stability of attenuated vaccines. In addition, the model will be extended to include various intracellular pathways and intercellular signaling which will enable further medical studies such as sepsis.

Our novel HPC-based simulation approach is a first step towards achieving these goals. We discuss the challenges in designing and implementing extremely fine-grained models for complex biological systems, such as writing reversible model code for databases implicit to the model, retraction of scheduled events and explicit diffusion of rare and unique particles. The attained scale of the simulation is unprecedented. Yet, another major scalability enhancement is necessary to enable many biological/medical studies.

VIII. Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (LDRD project 14-ERD-062 under Contract DE-AC52-07NA27344). LLNL-CONF-706818. This research used resources of the Oak Ridge Leadership Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] S. Bankes, "Exploratory Modeling for Policy Analysis," *Operations Research*, vol. 41, no. 3, pp. 435–449, 1993.
[2] A. Lauring and R. Andino, "Quasispecies Theory and the Behavior of RNA Viruses," *PLoS Pathogens*, vol. 6, no. 7, 2010.
[3] A. Khan *et al.*, "Conservation and Variability of Dengue Virus Proteins: Implications for Vaccine Design," *PLoS Neglected Tropical Diseases*, vol. 2, no. 8, August 2008.
[4] A. Zemla *et al.*, "GeneSV – an Approach to Help Characterize Possible Variations in Genomic and Protein Sequences," *Bioinformatics and Biology Insights*, vol. 8, Jan. 2014.
[5] R. Fujimoto, *Parallel and Distributed Simulation Systems*. John Wiley & Sons, 1999.
[6] D. Jefferson, "Virtual Time," *ACM Trans. on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, July 1985.
[7] C. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low-memory, modular Time Warp system," *J. Parallel Distrib. Comput.*, vol. 62, no. 11, 2002.
[8] E. Domingo and J. Holland, "RNA Virus Mutations and Fitness for Survival," *Annu. Rev. Microbiology*, vol. 51, pp. 151–178, Oct. 1997.
[9] S. Barua and S. Mitragotri, "Challenges Associated with Penetration of Nanoparticles across Cell and Tissue Barriers: A review of current status and future prospects," *NanoToday*, vol. 9, no. 2, April 2014.
[10] T. Chou, "Stochastic Entry of Enveloped Viruses: Fusion versus Endocytosis," *Biophys. J.*, vol. 93, no. 4, pp. 1116–1123, Aug. 2007.
[11] E. Simon-Loriere and E. Holmes, "Why do RNA viruses recombine?" *Nat. Rev. Microbiol.*, no. 9(8), July 2011.
[12] E. Asgari and M. Mofrad, "Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics," *PLoS ONE*, vol. 10, no. 11, Nov. 2015.
[13] P. Friebe and E. Harris, "Interplay of RNA Elements in the Dengue Virus 5' and 3' Ends Required for Viral RNA Replication," *J. Virol.*, vol. 84, no. 12, June 2010.
[14] D. Alvarez, C. Filomatori, and A. Gamarnik, "Functional Analysis of Dengue Virus Cyclization Sequences Located at the 5' and 3' UTRs," *Virology*, vol. 375, no. 1, 2008.
[15] S. Alcaraz-Estrada, M. Yocupicio-Monroy, and R. Angel, "Insights into Dengue Virus Genome Replication," *Future Virol.*, vol. 5, no. 5, pp. 575–592, 2010.
[16] K. Clyde, J. Barrera, and E. Harris, "The capsid-coding region hairpin element (cHP) is a critical determinant of dengue virus and West Nile virus RNA synthesis," *Virology*, vol. 379, no. 2, pp. 314–323, Sept. 2008.
[17] A. Khromykh, H. Meka, K. Guyatt, and E. Westaway, "Essential Role of Cyclization Sequences in Flavivirus RNA Replication," *J. Virol.*, vol. 75, no. 14, July 2001.
[18] C. Filomatori *et al.*, "A 5' RNA element promotes dengue virus RNA synthesis on a circular genome," *Genes and Development*, vol. 20, pp. 2238–2249, April 2006.
[19] ORNL, "Titan Specification," https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7.
[20] LLNL, "Vulcan Specification," https://computing.llnl.gov/?set=resources&page=OCF_resources#vulcan.
[21] Top500 List, https://www.top500.org/list/2016/11/?page=1.
[22] H. van der Schaar *et al.*, "Dissecting the Cell Entry Pathway of Dengue Virus by Single-Particle Tracking in Living Cells," *PLoS Pathogens*, vol. 4, no. 12, Dec. 2008.
[23] R. Milo, P. Jorgensen, U. Moran, G. Weber, and M. Springer, "BioNumbers—the database of key numbers in molecular and cell biology," *Nucleic Acids Res*, Jan. 2010, 38 (suppl 1): D750-D753.
[24] P. Barnes Jr. *et al.*, "Extreme-scale Optimistic Parallel Disrete Event Simulation with Dynamic Load Balancing," in *Winter Simul. Conf.*, 2014.
[25] E. Lazaro, C. Escarmis, E. Domingo, and S. Manrubia, "Modeling Viral Genome Fitness Evolution Associated with Serial Bottleneck Events: Evidence of Stationary States of Fitness," *J. Virol.*, vol. 76, no. 17, pp. 8675–8681, 2002.
[26] G. Jenkins, M. Worobey, C. Woelk, and E. Holmes, "Evidence for the Non-quasispecies Evolution of RNA Viruses," *Mol. Biol. Evol.*, vol. 18, no. 6, pp. 987–994, 2001.
[27] R. Ribeiro *et al.*, "Quantifying the Diversification of Hepatitis C Virus (HCV) during Primary Infection: Estimates of the In Vivo Mutation Rate," *PLoS Pathogens*, vol. 8, no. 8, 2012.
[28] P. Nonacs and K. Kapheim, "Modeling Disease Evolution with Multilevel Selection: HIV as a Quasispecies Social Genome," *J. Evolutionary Medicine*, vol. 1, 2012.
[29] V. Mozhayskiy, B. Miller, K. Ma, and I. Tagkopoulos, "A Scalable Multi-scale Framework for Parallel Simulation and Visualization of Microbial Evolution," in *TeraGrid Conf.: Extreme Digital Discovery*, 2011, pp. 7:1–7:8.
[30] Z. Xia, P. Das, T. Huynh, A. Royyuru, and R. Zhou, "Modeling Mutations of Influenza Virus with IBM Blue Gene," *IBM J. Res. Dev.*, vol. 55, no. 5, pp. 7:1–7:11, Sept. 2011.
[31] J. Da Silva and A. Hughes, "Monte Carlo Simulation of HIV-1 Evolution in Response to Selection by Antibodies," in *IEEE High Performance Computational Biology*, Aug. 2002.
[32] H. Woo and J. Reifman, "Quantitative Modeling of Virus Evolutionary Dynamics and Adaptation in Serial Passages Using Empirically Inferred Fitness Landscapes," *J. Virol.*, vol. 88, no. 2, pp. 1039–1050, Jan. 2014.