Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
88(7) 768–783

DOI: 10.1177/0037549711413001
sim.sagepub.com

Discrete event modeling and massively parallel execution of epidemic outbreak phenomena

Kalyan S Perumalla and Sudip K Seal

Abstract

In complex phenomena such as epidemiological outbreaks, the intensity of inherent feedback effects and the significant role of transients in the dynamics make simulation the only effective method for proactive, reactive or *post facto* analysis. The spatial scale, runtime speed, and behavioral detail needed in detailed simulations of epidemic outbreaks cannot be supported by sequential or small-scale parallel execution, making it necessary to use large-scale parallel processing. Here, an optimistic parallel execution of a new discrete event formulation of a reaction–diffusion simulation model of epidemic propagation is presented to facilitate a dramatic increase in the fidelity and speed by which epidemiological simulations can be performed. Rollback support needed during optimistic parallel execution is achieved by combining reverse computation with a small amount of incremental state saving. Parallel speedup of over 5,500 and other runtime performance metrics of the system are observed with weak-scaling execution on a small (8,192-core) Blue Gene/P system, while scalability with a weak-scaling speedup of over 10,000 is demonstrated on 65,536 cores of a large Cray XT5 system. Scenarios representing large population sizes, with mobility and detailed state evolution modeled at the level of each individual, exceeding several hundreds of millions of individuals in the largest cases, are successfully exercised to verify model scalability.

Keywords

discrete event, epidemiology, high performance computing, reverse computation

1. Introduction

1.1. Motivation

The significance of gaining better insights into the dynamics of large-scale epidemics is well known. The enormity of epidemic outbreak effects and the worldwide attention to controlling them are common knowledge, often appearing in daily news. In addition to the non-technical factors that come into play in the process of effectively dealing with epidemics, an important technical aspect continues to be elusive and remains to be explored, namely, gaining a good understanding of epidemic dynamics and the ways and means by which various contributing factors affect the propagation phenomena. Public health planners and policy makers use epidemiological simulations to study a variety of factors that influence epidemic dynamics within a population. For example, recently, Gojovic et al.[1] reported the use of such simulations to demonstrate the effect of timely delivery of vaccinations on the attack rate of H1N1.

Aside from analytical models based on simplifications, simulation continues to be an important tool. In contrast to numerical integration-based analysis of analytical (differential equation-based) epidemic models, simulation often provides flexibility in incorporating many factors. Large spatial scales and high behavioral detail contribute to the challenge of sustaining simulations of epidemic propagation dynamics at the scale of cities, states, and countries. Certain

Oak Ridge National Laboratory, Oak Ridge, TN, USA.

Corresponding author:
Kalyan S Perumalla, Oak Ridge National Laboratory, PO Box 2008, MS 6085, Oak Ridge, TN 37831-6085, USA
Email: perumallaks@ornl.gov

epidemics with global spans may even serve to motivate simulations at the world-scale.

Ideally, epidemiological models should be detailed enough to capture realistic models of the underlying phenomena and produce actionable insights. Realistic models tend to be very complex and the associated parameter space very large. In turn, this implies that the resulting computational problem becomes very large and unsuitable for sequential execution.

Decisions and policies are typically based on statistical inferences from results of multiple simulation runs that attempt to explore the model's associated parameter space as exhaustively as possible. This requires very fast turnaround times for each run so that enough statistics can be gathered within a reasonable duration of wall-clock time to base actionable decisions on. In light of the above, the need for parallel execution of such epidemiological models becomes evident and, not surprisingly, large-scale computational epidemiology has become an area of active research in recent years (see Section 1.4), particularly in an era when larger and more powerful parallel platforms are becoming increasingly common. Scalable algorithms are therefore imperative for large-scale realistic epidemiological simulations that can exploit the computing resources offered by today's state-of-the-art parallel platforms.

1.2. Simulation technology focus

The focus of this article is on advancing the state of the art in simulation technology to enable a dramatic leap in the population size, phenomenological process complexity (or model fidelity), and runtime speed for next-generation epidemic outbreak simulations. A greater emphasis is on advancing the simulation science and applying parallel computing methodologies than on exploring the domain science. Our goal is to enable domain experts in epidemiology to be armed with new, more powerful simulators to help advance the science of epidemics analysis. Specifically, we present a simulation development and scaling effort, based on a novel reversible parallel discrete event formulation, of a large class of epidemics that can be modeled using reaction–diffusion dynamics. We focus on the computational problem of sustaining simulations of large-scale epidemic propagation scenarios, and examine the performance effects of implementation on very large computational platforms containing 10^3–10^5 processor cores.

We develop a new parallel discrete event formulation of the reaction–diffusion model that is directly motivated by that reported by Barrett et al.,[2] while retaining the power and flexibility afforded by their individual models, which are nearly agent-based in modeling power (with parameters tunable down to each individual). We apply optimistic parallel execution techniques with a view to accommodating the low-lookahead conditions that may be present in certain scenarios. For rollback, we employ a combination of state saving and reverse computation. For scalability, we utilize supercomputing platforms to increase the speed of simulation of large-scale scenarios. The goals of our effort are not only to provide parallel discrete event simulation (PDES) alternatives to the currently best-known results that are non-PDES-based, but also to achieve new levels of scale (hundreds of millions of individuals) simulated on many thousands of processor cores.

1.3. Contributions

This article makes multiple contributions. First, our new PDES-based formulation of reaction–diffusion models (presented in greater detail later in the article), is unique in that host mobility, intra-host state evolution, and inter-host reactions are all modeled in full, discrete event fashion. The time scales are completely decoupled from each other, and do not require any *a priori* determination of a time step and/or a minimum time increment, unlike the prior formulation of Barrett et al.[2] Our model also enhances the mobility model of Barrett et al.[2] by introducing arbitrary travel time delays across locations. In the context of optimistic simulation, our reversible model for epidemics is unique. In terms of scalability, the results reported here constitute the most scalable PDES execution of epidemic models. Also, to the best of our knowledge, the largest run of the simulation scenario on 65,536 processor cores of Cray XT5 reported in Section 4 is the largest processor count to date of any non-trivial PDES application reported in the literature.

1.4. Related work

The epidemic modeling literature is vast, from classical differential equations that appeared in the 1920s for aggregate phenomena such as critical thresholds and herd immunity, to articles appearing as recently as 2009 on phenomena such as the emergence of critical thresholds on a variety of small-world and scale-free social networks.[3–7] In the past few years, agent-based simulations were used by the National Institutes of Health's Models of Infectious Disease Agent Study (MIDAS) group to shape avian flu policy.[8,9] Other methodologies such as patch models, distance-transmission models and multi-group models are discussed by Riley.[10] A new class of reaction–diffusion-based disaggregate models[11,12] has emerged in the past few years, to facilitate planning by federal agencies (DHS, DoD and NIH).

Although, historically, epidemic simulations are sequentially executed, parallel simulation has been used in the late 1990s and mid 2000s to accommodate increasingly larger sizes of epidemic networks, and also speed up the execution to meet real-time constraints. Within parallel simulations, time-stepped approaches have been used to execute certain complex models at large-scale. The SPaSM simulator[13] is an example of such a scalable framework that has been successfully used for analyzing very large scenarios, including certain country-scale pandemic flu propagation analyses. In scenarios involving highly varying time scales and non-uniform inter-person interaction structures, however, discrete event simulation can offer faster advances in simulation time. On the other hand, PDES offers its own challenges: the need to reformulate the dynamics in terms of discrete events, and the need to reduce runtime overheads to make it possible to scale execution to large numbers of processors. The most closely related work is an algorithm of Parker[14] that uses a 'correction method' on state updates that are processed out of order. However, it is not formulated in traditional PDES concepts of logical processes, events, and rollbacks, and hence hard to compare with PDES techniques that are more generally applicable. It is restricted to shared memory machines, and has been used to simulate millions of lightweight Java-based agents, exploiting a simplified state machine of an individual's infection states, and hence does not generalize to more complex scenarios, models, and experimentation platforms.

A city-scale small-pox spread simulation was reported by Linebarger et al.,[15] with the focus more on interoperability of simulator modules, less on scalability and efficiency. Other, peripherally related, work includes Monte Carlo simulation models.[16]

Systems such as EpiSimdemics, EpiSims, and EpiFast from Virginia Tech[2,17] have advanced the state of the art in epidemics analyses greatly in the past few years. We have been directly motivated by an earlier work of Barrett et al.[2] that reported scalability of execution on up to 512 processor cores, encountering certain synchronization problems beyond that scale. An enhanced version of this work to include models of exogenous interventions[17] also scales to about 250 processors.

Our current work borrows this reaction–diffusion model, but explores an alternative execution method distinct from the parallelization approach that can be roughly classified as functional parallelism. We apply the latest PDES methodologies to explore the potential of PDES executions to well beyond a thousand processor cores. This article is an extended version of our earlier conference paper,[18] containing additional model detail, cost analysis, and performance results.

1.5. Organization

The rest of the document is organized as follows. The forward model is described in Section 2. The reverse model is described in Section 3. An analysis of average memory size with state saving and reverse computation is presented in Section 4. The implementation of the system, the characteristics of the experimentation platform, and the specification of the benchmarks are described in Section 5, followed by a detailed performance study in Section 6. The article is summarized and ongoing work identified in Section 7.

2. Forward execution model

The model and the details of the reaction–diffusion system are described in this section, along with the discrete event execution formulation.

2.1. Model based on reaction–diffusion

A large class of epidemics may be modeled using a combination of reaction and diffusion processes. The reactions arise as a result of inter-entity interactions (e.g. physical proximity for influenza). The dynamic chances for interactions arise as a result of the mobility of the entities (e.g. meetings and other co-located activities). Geographical diffusion of entities facilitates chances for interactions among varying sample sets of entities.

Figure 1 shows a schematic for the reaction–diffusion abstraction on a system of individuals. Interaction points called 'locations' demarcate the extents of

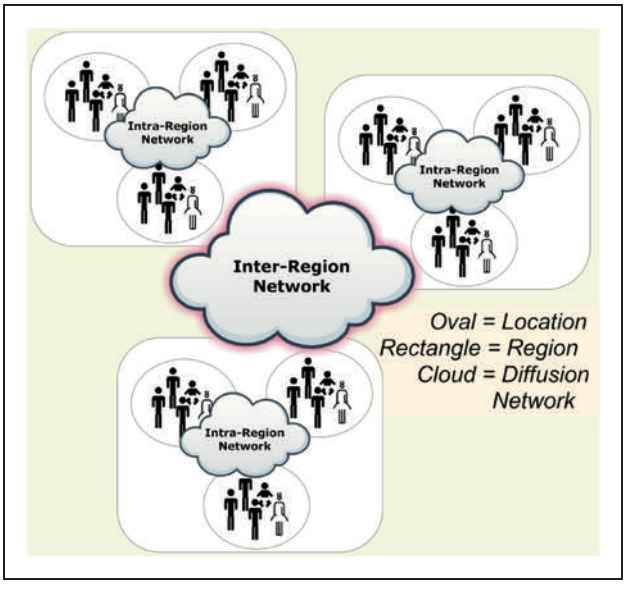

Figure 1. Abstraction of the system for epidemic propagation.

interaction that individuals within that location incur during the time they are present at that location. Multiple locations are contained within a region. An arbitrary diffusion network may be defined among locations within a region. Similarly, an arbitrary diffusion network may be defined among regions. The diffusion network can be dynamic in nature; in other words, the connectivity graph may be changed at the modeler's/user's will at runtime, as needed by any specific scenario being modeled. Latencies of activities (interaction and/or mobility) along the links of the network may be arbitrarily specified.

Reaction–diffusion-based models typically consist of: (a) a set of interacting entities, such as humans; (b) a network graph that represents the interaction structure of the population under study; (c) a reaction process, called within-host progression, that models the evolution of the disease within an individual entity in the population; and (d) a diffusion process, called between-host transmission, that models the transmission of the disease between individual entities. Other sub-models may be added to accommodate additional aspects such as resource limitations on all processes.

A reaction function defines the infection behavior of co-located individuals (probability of an uninfected entity getting infected or, alternatively, an infectious individual infecting other susceptible population).

The within-host progression of the infection is modeled by a probabilistic timed transition system (PTTS) which is a finite state machine with probabilistic, timed state transitions. A PTTS reflects the reaction that changes the state of an individual within the population. We use the PTTS framework of Barrett et al.[2] unmodified.

The reaction function also includes the determination of the probability of transmission within the group of individuals at any location. For an individual i, the between-host transmission is modeled by the probability function p_i, borrowed unmodified from Barrett et al.[2] and reproduced here for convenience:

$$p_i = 1 - e^{t\sum_{r\in R} N_r ln(1-rs_i\rho)} \quad (1)$$

In this equation, τ is the duration of exposure, R is the set of infectivities of the infected individuals, N_r is the number of infectious individuals with infectivity r, s_i is the susceptibility of i and ρ is the transmissibility which is a disease specific property. The rationale and additional detail on these variables are documented by Barrett et al.[2]

The underlying social network consists of the following entities defined hierarchically as: (a) individual person; (b) location, a set of individuals; (c) region, a set of locations; and (d) domain, a set of regions. The scale of the system depends on the definitions that we associate with the above entities. For example, a state level simulation can associate locations with office buildings or shopping malls, regions can be associated with subdivisions, towns or cities and the state itself as a group of regions which represents the computational domain over a state-level social network. Similarly, a country-level simulation can be appropriately redefined.

In summary, the system consists of (a) individual entities each with its own characteristics, (b) locations that hold individuals for periods of time, (c) regions that represent a set of logically related locations, who have a diffusion network among themselves, (d) a reaction function that specifies the probabilities of infections over time within a location, and (e) a two-level diffusion function that takes an individual from a location–region pair to another location–region pair.

In a simplified network, the following parameters may be defined: (i) *infected probability*: percentage of population infected at time $t = 0$; (ii) *vaccinated probability*: percentage of population vaccinated at time $t = 0$; (iii) *mean stay time*: average time that an individual stays within a location; (iv) *mean local travel time*: average time taken by an individual to travel between two locations within a region; (v) *mean remote travel time*: average time taken by an individual to travel between locations across two regions; and (vi) *locality probability*: probability that an individual stays within a region.

2.2. Relation to real-world entities

The abstraction can be used to model activity at multiple spatial scales. If the number of persons per location is S_l and the number of locations per region is L_r, then Table 1 illustrates the rough scales of population for which epidemic outbreaks could be analyzed using this model.

The model is sufficiently flexible to accommodate intra-city and inter-city trips (work, stores, homes, schools), and varying levels of exposure times for reactions. In the largest case, one can envision modeling inter-country transport (ship, air), in combination

Table 1. Example scales of systems

S_l	L_r	*Scenario*
10	100	Small community
100	100	Small town
1000	100	Small city
1000	1000	Large city
10000	100	Rural state

with intra-country (inter-state) diffusion (car, air). A visualization of a motivating example scenario is shown in the appendix, using a mapping of the modeling units (locations and regions) to a few cities and states of the USA. Trip statistics obtained from the Bureau of Transportation Statistics are typically utilized to initialize the network structure and time distributions.

2.3. Modeling operations and policies

The same abstract model is in fact sufficiently powerful to model complex operations and policies. For example, curfews can be modeled by using an outgoing probability of unity and incoming probability of zero for the location corresponding to the curfew (e.g. a public park). Similarly, an outgoing probability of zero and incoming probability of unity may be used in the diffusion network to model quarantined locations (e.g. a school or hospital).

Analogously, it is possible to incorporate vaccination production delays (manufacturing latencies, vehicular transportation times) by setting or varying diffusion network latencies accordingly.

2.4. Our discrete event model

As mentioned previously, we employ a purely discrete event style of evolution for all aspects of the model. Each location is mapped to a single logical process (LP). A number of locations (specified by the user at runtime) constitute a region, and each region is mapped to a processor. The location LP processes three types of events, namely: (a) arrival event (*AE*); (b) state change event (*SCE*); and (c) departure event (*DE*). Each individual person in a location is assigned a globally unique integer identity to distinguish it within the entire population. A customized PTTS (specifiable by the user on an individual basis at initialization) is used to model the within-host progression of the infection. The state composition of each person and each location (LP) are given in Figures 2 and 3, respectively.

The forward execution algorithm is shown in Figure 4. When an individual arrives, an *AE* is

Person State

ptype: Person Type and/or Traits

pid: Global Identifier

seed: RNG Seed

istate: Infection State

Figure 2. State encapsulated by every person.

Location Data Structures (LP State)

occ: Occupant Set of Persons (Person States)

omap: Map from ID to Person in *occ* Set

gptts: Default PTTS across local occupants

Figure 3. Data structures within each location logical process (LP).

Forward Algorithm

If(Event is *AE*)
  Insert *AE.Person* into *Occupant Set*
  Incorporate *AE.Person*'s random seed locally
  dt=Randomized depart time of *AE.Person*
  Schedule *DE* for *AE.Person* at *now+dt*
If(*AE.Person* turned infectious between now and its departure time at source)
  Progress infection state of *AE.Person* *(schedule SCE for AE.Person)*
End If
  Compute Equation (1) (reaction function)
  Determine new infections due to Equation (1)
For each person *P* in *occ* newly infected now
  Progress infection state of *P* by PTTS *(retract previous SCE of P and schedule a new SCE for P)*
End For
Else If(Event is *DE*)
  Locate person *P* in *occ* with identifier *DE.pid*
  Remove *P* from *occ*
  Determine destination *L* (diffusion function)
  If *P* has local pending *SCE*, note residual δt
  Determine travel time *dt* (diffusion function)
  Schedule *AE* for *P* to *L* at *now+dt*
Else /*Event is SCE*/
  Locate person *P* in *occ* with ID *SCE.pid*
  Progress infection state of *P* by PTTS *(schedule SCE for P)*
If(*P* just turned infectious)
  Compute Equation (1) (reaction function)
  Determine new infections due to Equation (1)
For each person *Q* in *occ* newly infected
  Progress infection state of *Q* by PTTS
  (retract previous SCE of Q and schedule a new SCE for Q)
End For
End If
End If

Figure 4. Forward event processing at each logical process.

triggered in that location. As part of processing an *AE*, the incoming person is inserted into an occupant set. If the incoming individual arrives in an infectious state, its transmission within the local population is computed using Equation (1). If it is uninfected, then its within-host progression is computed using the PTTS. The incoming person's departure is scheduled based on the parameter *mean stay time*. The individual's next departure time is computed, and departure event *DE* is scheduled by the location to itself as reminder to eject the individual from the location's state at that time. At the time the departure event is processed, a destination location (either local to the region or in another region across processors) is selected, the travel time computed, and the state of the person corresponding to the moment of departure at the source location is packed into the event and sent.

The processing of a *DE* marks the exit of that person from its current location. The outgoing individual is removed from the occupant set and an *AE* is scheduled at the destination location.

When an individual *P* within a location changes its state of infection, an *SCE* is triggered. The infection state of *P* is then evolved using its PTTS. If the final state is infectious, then its effect on the rest of the occupants within the location is computed using Equation (1).

A couple of important nuances must be considered for correctness: (a) a person turning infectious while in transit must initiate infections as appropriate on arrival at the destination; and (b) random number stream continuity must be maintained across processors when persons cross processors. The first nuance is taken into account by noting the residual time of infection dormancy in *AE* so that the receiving location can verify the special condition and act accordingly (special case trapped in the forward algorithm by the first conditional in processing *AE*). The latter nuance is handled by shipping in *AE* the random number generator seed of that person along with the person's state. The receiving processor restarts its random number stream accurately at the received starting seed for that person.

3. Reverse execution model

We now turn to the reverse execution handlers that make use of the information generated in the modified forward execution to undo forward operation.

3.1. Modified LP data structures

Since the original forward-only model is not reversible as is, a few variables need to be added to achieve reversibility. Note that this does not imply the use of state saving; it only implies the increase of sequential processing memory need by a constant factor. In other words, no state *log* is accumulated as a result of this augmentation, but only the original copy of the (single) state is increased by a *constant* amount.

The variables added to each person are shown in Figure 5. The *prev_istate* is used to mark its previous infection state to remember an irrecoverable previous state, if any, in the PTTS, encountered by the person. The *infect_ts* is added to disambiguate the identity, if any, of event that triggered the infection of this person.

Figure 6. shows the addition of a 'departed' set to each location to accommodate reversibility of *DE*. This is used to 'keep the person around' in case the departure event is rolled back. Again, this does not constitute state saving by itself. To compare, even if the best (incremental) state-saving techniques are employed, two copies of a person's state would be generated in the incremental state log for every move from occupant set to departed set, bringing the total memory to three person states, compared with the two states in our reverse computing scheme.

3.2. Modified forward event processing

The forward event processing is augmented as follows: (a) any time an infection is initiated on a person, its infection time is noted in the person's state; (b) departing persons in *DE* are moved to the 'departed set' rather than discarded; (c) previous state of a person is noted in the corresponding *SCE* event for every state change. Owing to space limitations, the modified forward algorithm is not reproduced here.

3.3. Backward event processing

The reverse execution by locations is shown in Figure 7.

The reversal of arrival event initiates the reversal of locally scheduled departure events, and undoes local infection state change event for the subject individual.

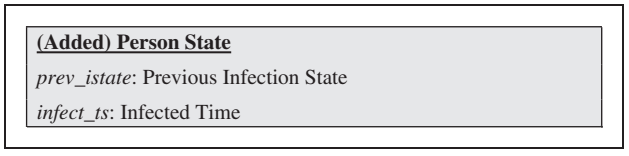

Figure 5. State added to person for reversibility.

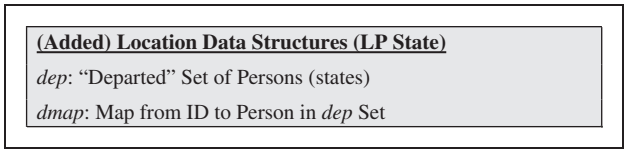

Figure 6. State Added to each location logical process (LP) for reversibility.

The departure set is maintained should a *DE* need to be reversed in the future. This is used to restore an optimistically departed person back into the occupant set.

Since, as part of processing an *SCE*, the previous infection state is stored, it is restored in backward execution. Same holds true for detecting the set of infected events when the arrival of an infectious person and/or a state change event causes one or more infections. The set of infected individuals is detected accurately by comparing their infection timestamps with the current simulation time.

3.4. Event commit operations

An important effect of the use of a departure set is that the size of the set accumulates over simulation time, since all departed persons are tentatively retained there. Thus, they need to be reclaimed. This is easily done by periodically flushing those entries that contain persons whose departure time is less than the global virtual time. Thus, an additional conditional is added to the event processing loop that detects this condition and the departure set is periodically flushed.

```
Backward Algorithm
If(Event is AE)
 For each person P in occ infected at now
    Restore its states to previous in PTTS
    Reverse its random number generator
 End For
    Remove AE.Person from occ
Else If(Event is DE)
    Locate person P in dep with identifier DE.pid
    Remove P from dep
    Add P to occ
Else /*Event is SCE*/
    Locate person P in occ with ID SCE.pid
 If( P is infectious)
For each person Q in occ infected now
    Restore its states to previous in PTTS
    Reverse its random number generator
End For
 End If
    Restore state of P to previous in PTTS
    Reverse its random number generator
End If
```

Figure 7. Backward event processing at each logical process.

3.5. Random number generation

We use the well-known technique of reversible random number generation[19] to undo the several random number generation calls that appear in the model. These appear in the state transition function for PTTS, the computation of infection probability in the reaction function, and the selection of time and destination in the mobility determination (diffusion).

4. Analysis of state memory size

To decide on the type of undo method to employ for restoring the state of a location upon a rollback, we analyze the theoretical memory needs using some of the well-known methods. We examine some of the popular methods that use state-saving, reverse computation or a combination. Table 2 lists the average memory needs for each approach: Copy State Saving (CSS), Incremental State Saving (ISS), Reverse Computation with some State Saving (RC+SS), and Pure Reverse Computation (PRC).

The parameters used in the analysis are: μ_{occ} is the average number of occupants (persons) at a location when an event of a specific type is executed, μ_{inf} is the average number of infections initiated at a location by an arrival of a person who newly turned infectious, R is the memory size of each person's state, G is the memory size of each random number generator seed, D is the memory size needed to represent a state in the PTTS, TS is the memory size occupied by a timestamp value.

Let us consider the easiest to analyze, namely, CSS. The memory cost for CSS is straightforward: before executing any event, the entire state of the LP is saved. The state size of each location is equal to the sum of state sizes of all resident persons in the location's occupant set. Given that the average number of

Table 2. Average memory size for rollback per event using state saving, reverse computation or combination

Event type	CSS	ISS	RC+ISS	PRC
Arrival	$\mu_{occ}\times R$	$\mu_{inf}\times(G+D)$	$\mu_{inf}\times(TS+D)$	0
Infection state change	$\mu_{occ}\times R$	$G+D$	D	0
Departure	$\mu_{occ}\times R$	$[\varepsilon_{::}2R]$	0	0

persons in the occupant set is μ_{occ}, the size is $\mu_{occ} \times R$ which remains the same for all events.

The next improved method is the well-known ISS that saves a copy of only the portions of the state that are modified during event execution. For an arrival event, the state modifications include the set of resident persons that are infected by an infectious arrival. Each resident person who is newly infected by an infectious arrival generates a new random number to determine its next state in the PTTS and then updates its new infection (PTTS) state. Thus, each infected resident incurs an incremental update to its state of size $G+D$, giving a total incremental state cost of $\mu_{inf} \times (G+D)$ per arrival. A change in infection (PTTS) state performed by an SCE type of event inflicts a similar incremental cost of $(G+D)$ for a resident. A departure event moves the person from occupant set to the departed set, giving a memory cost of anywhere between ε and $2R$, depending on how the sets are implemented. A highly optimized implementation may incur a cost of only a small number ε of bits by reorganizing a small number of pointers, while a naïve implementation may delete the state from the occupant set and write new data in the departed set, giving a cost of $2R$ to undo such a move.

The combination of reverse computation and state saving as employed in our scheme incurs a reduced costs for all three events. Upon infection of any person by an infectious arrival, the previous state of that person is saved along with the timestamp of the infection, giving a total of $\mu_{inf} \times (TS+D)$ per arrival event. The reversibility of random number generators[19] obviates the memory cost of saving the seed. Similarly, an infection change event only incurs the cost of saving the previous infection state. A departure event simply moves the person from occupant set to departed set, which is perfectly reversible, thereby incurring zero memory cost.

It is very important to note that the memory cost associated with infectious arrivals as well as that for infection state change (SCE) events can be entirely avoided with pure reverse computation. This can be achieved by temporarily reversing the random number generator stream sufficiently far in the past to disambiguate the previous state from which the current state was reached. Note that this is always possible with all PTTS machines in which recovered states do not loop back to the (uninfected) initial state. However, owing to the higher runtime cost and software implementation complexity associated with the computation needed to avoid the memory for perfect reversibility of infection change events, the RC+ISS method serves as a good trade-off between memory and runtime costs.

Note also that, in realistic scenarios, the number of arrival and departure events (AE and DE) is significantly larger than infection state change events (SCE). In the same scenarios, average number of infections μ_{inf} is also extremely low, tending to zero. In such cases, the memory cost of RC+ISS asymptotically becomes negligible. Owing to these considerations, we chose the RC+ISS approach to support state restoration upon rollback. It is indeed conceivable for another implementation to exclusively use ISS; however, its memory cost would be necessarily bounded on the lower side by that of RC+ISS, as seen in Table 2.

5. Implementation and benchmarks

The application was developed in C++ on top of a simulation engine that supports the concepts of LPs, events for exchanging time-stamped messages among logical processes and virtual time-synchronized delivery of events to LPs. The engine avoids all use of collective communication calls and implements a variant of global virtual time synchronization that is purely asynchronous in nature. It avoids blocking in all places and is built with scalable reduction algorithms that are realized in user space using point-to-point, non-blocking communications.

5.1. Comparison with traditional models

Traditional epidemiological models, such as the SIR model and several such variants, are based on simplified views of the population under study. In the SIR model, each member of the population progresses from being in a susceptible (S) state to an infectious (I) state and finally to a recovered (R) state. The total number of individuals in each of the states is a time-dependent function. Typically, $S(t)$, $I(t)$, and $R(t)$ are related to each other through coupled differential equations, under the aggregate constraint that $S(t) + I(t) + R(t) = N$ which preserves the total population size. The dynamics evolve through predefined infection and recovery rates that connect the S and I states, and the I and R states, respectively. In contrast to such aggregate models, the dynamics of the reaction–diffusion-based model adopted in this paper (borrowed from recent literature[2]) evolve through probabilistic functions. The reaction and diffusion dynamics are governed by probabilistic timed state transitions and the probability function given by Equation (1).

It is important, however, for a higher fidelity model to be able to duplicate the dynamics of the traditional aggregate models in scenarios that represent aggregate dynamics. A verification of such a capability, to duplicate the logistic curve generally followed by aggregate model dynamics, is performed

on our system. In Figure 8, the cumulative number of infections in a population size of 1,000 individuals is plotted using the parameter set [**a a a**] (see Section 5.3) with an initial population distribution of only one individual per location.

Note that, in practice, epidemiological outbreak simulations of interest are rarely simulated to the point where such large fractions of population are infected. This verification of dynamics following the logistic curve is only performed as one of the tests to assure the implementation's correctness properties.

5.2. Hardware and software platforms

The simulations were run on Cray XT4, Cray XT5, and Blue Gene/P machines at the National Center for Computational Sciences (see http://www.nccs.gov). The XT4 contains 7,832 compute nodes, each node containing a quad-core 2.1 GHz AMD Opteron processor with 8 GB of memory. The nodes are connected via a high-bandwidth SeaStar interconnect. Internally, the MPI implementation is based on Cray's implementation of Portals 3.3 messaging interface. At the time when experiments were run on the XT5, it contained 36,864 quad-core AMD Opteron Budapest processors with 2 GB of memory per core. The nodes are connected by a SeaStar-II interconnect. The Blue Gene/P that was used is a 27 TF system consisting of 2,048 850 MHz IBM quad-core 450d PowerPC processors and 2 GB of memory per node.

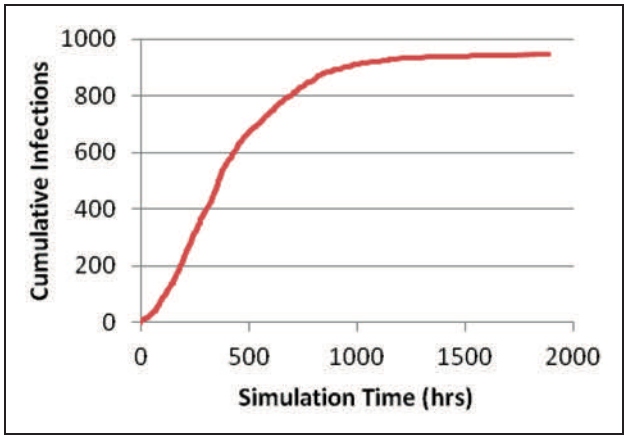

Figure 8. Verification of propagation dynamics in a scenario (with one location per region, one individual per location) that approximates the high-fidelity model with traditional aggregate models such as SIR.

5.3. Benchmark scenario description

The performance of the application is studied with respect to changes in: (a) population distribution; (b) reaction–diffusion parameters; and (c) mobility parameters. Several combinations of the above model parameters were tested. A few of these scenarios are described next.

5.3.1. Population. Two different population distributions, labeled by **I** and **II**, were explored. For the same total population, the number of individuals per location (LP) in distribution **I** is one order of magnitude higher than in distribution **II**.

5.3.2. Reaction–diffusion. Three different assignments of the reaction parameters are used, each, labeled **a**, **b** and **c**, were chosen. Two mobility distributions are chosen, labeled **a** and **b**. Their qualitative descriptions are provided in Table 3, and their actual numeric values are listed in Table 4.

5.3.2. Scenarios. The experimental scenarios, thus, are represented by a cross product of the population distribution, reaction parameters, and diffusion parameters. For example, a scenario is completely determined by the choice of the population set (number of locations per region, and initial number of persons per location), the choice of set for intra-person reaction parameters (set **a** or **b** or **c**), the choice of inter-person reaction parameters (set **a**, **b** or **c**), and the choice of diffusion parameters (set **a** or **b**). For any population set, we define the scenario by the tuple $\langle R1, R2, D\rangle$ where $R1 \in \{a, b, c\}$, $R2 \in \{a, b, c\}$, and $D \in \{a, b\}$.

The intra-person reaction parameters define the infection characteristics of each individual. The inter-person reaction parameters are used to define the aggregate constants that define the reaction according to

Table 3. Qualitative description of the sets of parameters chosen for our experiments

Type	Tag	Set a	Set b
Reaction (intra-person)	R1	Level I	Level II
Reaction (inter-person)	R2	Infectivity is greater than susceptibility	Susceptibility is greater than infectivity
Diffusion (mobility)	D	High	Low

Table 4. Parameter sets used in the scenarios simulated in runtime performance experiments

	Set a	Set b	Set c
Reaction (R1)			
Normal-Uninfected-Latent	0.9	0.5	0.1
Normal-Latent-Infectious	1.0	1.0	1.0
Normal-Infectious-Recovered	1.0	1.0	1.0
Normal-Uninfected-Incubating	0.1	0.5	0.9
Normal-Incubating-Asympt	1.0	1.0	1.0
Normal-Asympt-Recovered	1.0	1.0	1.0
Normal-Recovered-Recovered	1.0	1.0	1.0
Vaccinated-Uninfected-Recovered	0.5	0.5	0.2
Vaccinated-Uninfected-Latent	0.2	0.3	0.4
Vaccinated-Latent-Infectious	1.0	1.0	1.0
Vaccinated-Infectious-Recovered	1.0	1.0	1.0
Vaccinated-Uninfected-Incubating	0.3	0.2	0.4
Vaccinated-Incubating-Asympt	1.0	1.0	1.0
Vaccinated-Asympt-Recovered	1.0	1.0	1.0
Vaccinated-Recovered-Recovered	1.0	1.0	1.0
Reaction (R2)			
Infectivity	1.0	0.5	1.0
Susceptibility	0.5	1.0	0.5
Transmissibility	5e-4	5e-4	5e-4
Diffusion (D)			
Locality (%)	75	90	
Lookahead (h)	0.3	0.5	
Mean stay time (h)	8.0	5.0	
Mean local travel time (h)	0.5	1.0	
Mean remote travel time (h)	8.0	12	

Equation (1). The diffusion parameters define the mobility characteristics that directly influence the types and intensity of mixing among individuals across locations and regions, thereby affecting their opportunities for infection.

The values chosen are only representative for the range of runtime dynamics we desire to exercise, since the infection and mobility dynamics have a noticeable bearing on the runtime performance.

6. Performance study

We now present the performance study based on the experiments with the benchmark scenarios on Blue Gene/P and Cray XT4/XT5 platforms.

The majority of runs to cover the benchmark scenarios have been executed on the 8,192-core Blue Gene/P. Owing to allocation limits on computing time, a smaller number of scenarios were executed on the larger 32,768-core Cray XT4, and only a single scenario was executed on the largest Cray XT5 that has over 200,000 cores.

The parallel speedup obtained on Blue Gene/P is shown for different scenarios in Figures 9–12. Rollback statistics for a few of the representative runs on the Blue Gene/P are shown in Figure 13. Parallel speedups for Blue Gene/P in these figures are plotted with respect to a baseline execution on 128 cores which is the smallest number of processor cores on the Blue Gene/P platform that can be used for any parallel execution. The speedup on 128 cores is, accordingly, defined as the baseline speedup with 100% efficiency.

Note that, in the largest configurations executed on the Blue Gene/P, over 800 million individuals are represented (1,000 person/location, 10 locations/region, 8,192 regions). This is a stress test on the hardware capacity of the system, and hence, as seen in Figure 12, only a few configurations of reaction and diffusion parameters were successful at the largest population sizes on the Blue Gene/P. Blue Gene systems of higher scale are available elsewhere, which we intend to

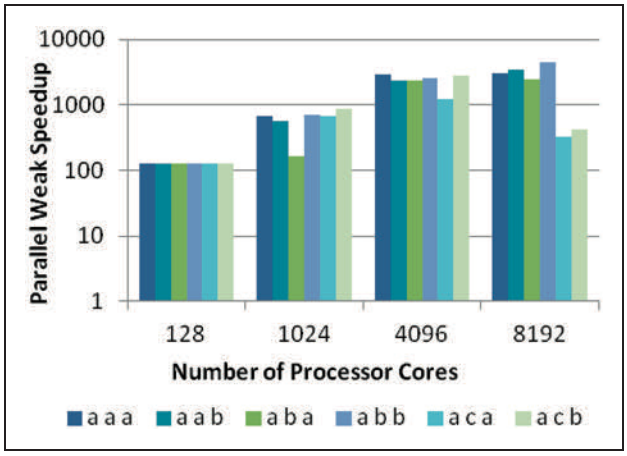

Figure 9. Parallel speedup on Blue Gene/P with 100 persons/location, 10 locations/region.

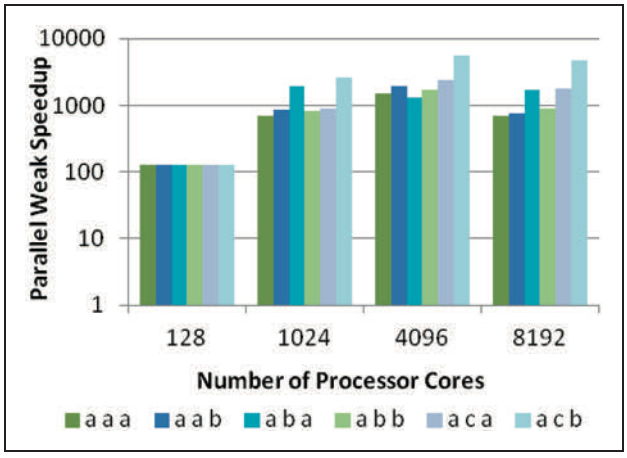

Figure 10. Parallel speedup on Blue Gene/P with 1,000 persons/location, 10 locations/region.

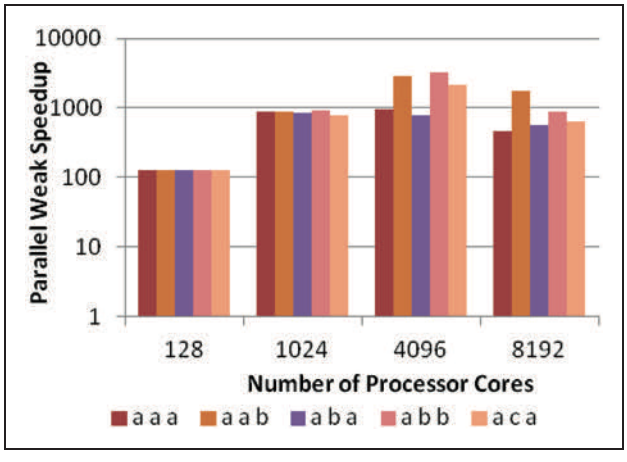

Figure 11. Parallel speedup on Blue Gene/P with 100 persons/location, 100 locations/region.

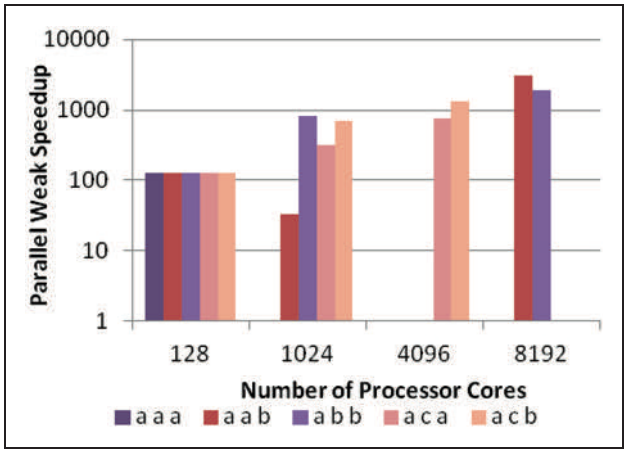

Figure 12. Parallel speedup on Blue Gene/P with 1,000 persons/location, 100 locations/region.

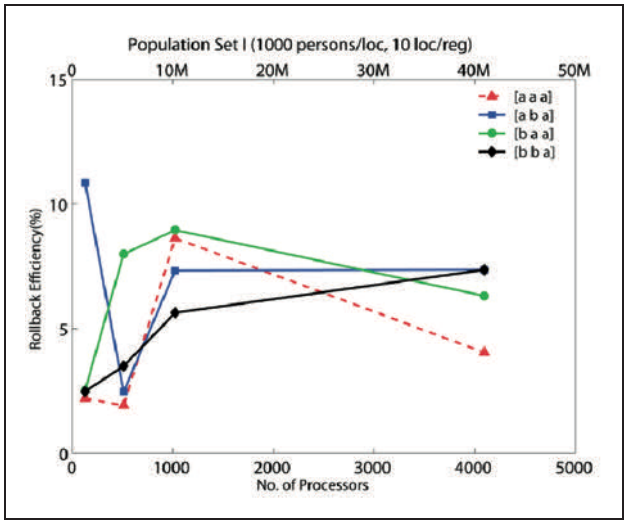

Figure 13. Percentage of events rolled back (rollback efficiency) on Blue Gene/P.

access in the future to test scaling beyond 8,192 cores on the Blue Gene architecture.

The results from the Cray XT4 are shown in Figure 14 and 15. The largest Cray XT5 results are shown in Figure 16.

6.1. Analysis

One of the effects of population distribution on the parallel performance can be observed in the results obtained on Blue Gene/P. They show the relative speedup for different population sets on processor counts varying from 128 to 8,192 on the Blue Gene/P architecture.

The probability that an individual chooses a destination location that is resident on a remote region is the same as that of choosing one that is locally available,

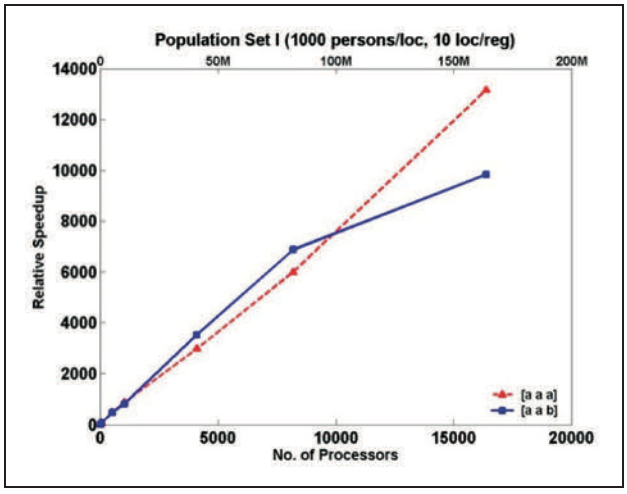

Figure 14. Speedup on Cray XT4 for population set I.

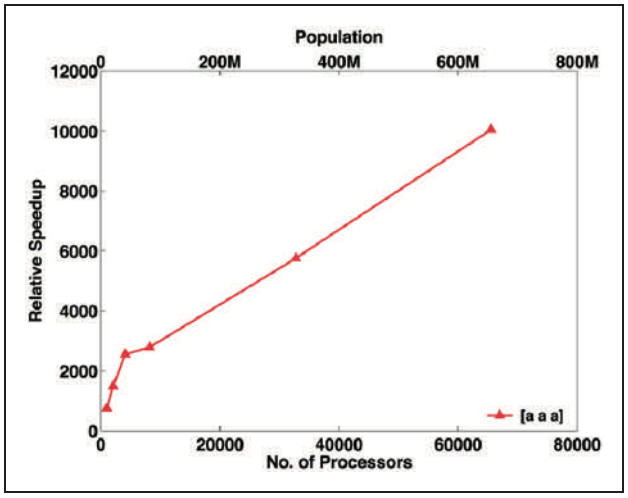

Figure 16. Speedup on Cray XT5.

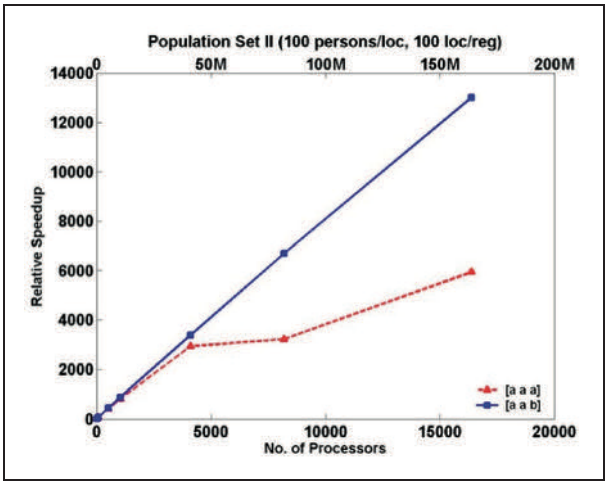

Figure 15. Speedup on Cray XT4 for population set II, with the same total population as in Figure 9.

As a result, region boundaries are crossed more often when there are less destination locations available locally. Since regions are mapped to processors, the resulting communication overhead increases as the number of locally available locations decreases. This effect is evident in the Blue Gene/P results, which indicate the degradation of parallel performance in going from 100 to 10 locations per region for the same total population.

Degradation of speedup observed at the full system scale of 8,192 cores of the Blue Gene/P can be attributed to operating system jitter that is well known to be detrimental to parallel performance at the full scale of any system. Significant speedup improvements are expected on the same number of cores but on a larger system; this is in fact evidenced by the fact that the same scenarios deliver good speedup on the larger XT4 platform.

Experiments were carried out on the Cray XT4 with simulation runs carried out on up to 16,384 processors. In Figure 14, only the diffusion parameter set is varied while in Figure 15 only the mobility parameter set is varied. For the same parameter set **[a a b]**, the parallel performance is seen to improve in Figure 15 with an increase in the number of locations that are locally available, as explained above. This is not the case, however, for the scenario with parameter set **[a a a]** (see Figure 15). This highlights the competing effects of mobility of individuals within a population and the distribution of the individuals within it.

Figure 13 indicates that for a variety of reaction–diffusion and mobility conditions, the percentage of events rolled back in the optimistic parallel simulation remains under control (well under 10% for all tested scenarios). Finally, Figure 16 demonstrates the scalability of the simulation to 65,536 processors on a Cray XT5 platform with 10,000× speedup.

As can be seen from the Blue Gene/P runs, runtime performance of the parallel simulation is not only dependent on the hardware characteristics of the computing platform but also closely related to the scenario that is being simulated. For example, the observed speedup on 1,024 cores approximately varies between 700 and 2,500 with scenarios containing 1,000 persons/location and 10 locations/region (see Figure 10). Nearly 60% efficiency is observed on 8,192 processors for the scenario [**a c b**] with 1,000 persons/location and 10 locations/region. Super-linear speedup can be observed in a few instances, such as, scenario [**a c b**] on both 1,024

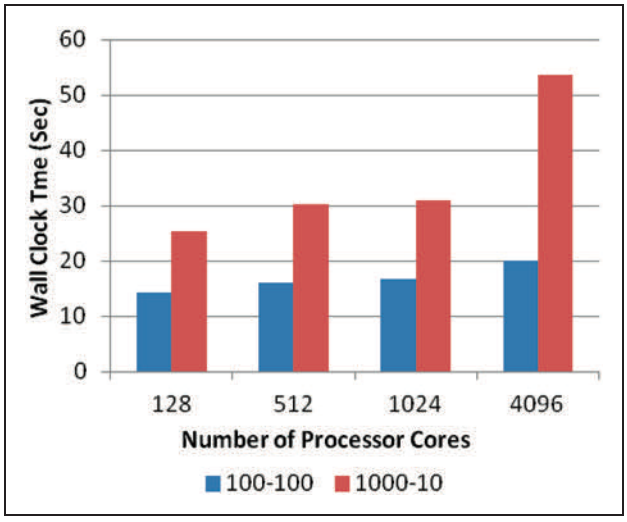

Figure 17. Simulation runtime for two different decompositions (100 persons/location with 100 locations/region, and 1,000 persons/location with 10 locations/region) of the same total population of 10,000 persons/region, for reaction–diffusion settings of [a a b].

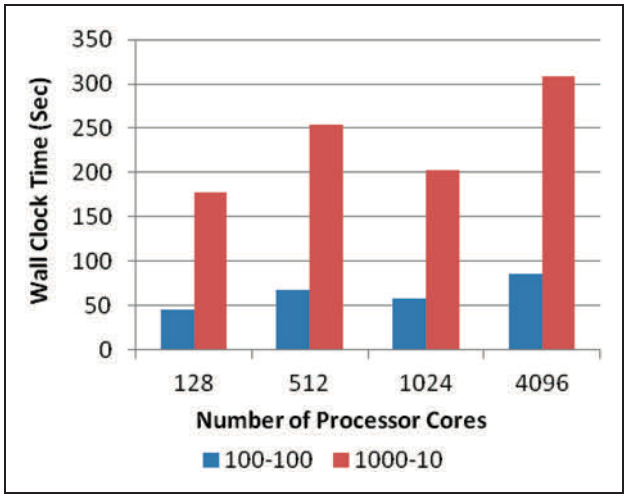

Figure 18. Simulation runtime for two different decompositions (100 persons/location with 100 locations/region, and 1,000 persons/location with 10 locations/region) of the same total population of 10,000 persons/region for reaction–diffusion settings of [a c a].

and 4,096 cores, and scenario [**a b a**] on 1,024 cores, both with 1,000 persons/location and 10 locations/region. One of the main reasons for such variance in the speedups for the same problem size and core-count is that the phenomenological dynamics can differ significantly from one scenario to another. For example, scenarios with smaller values of the locality parameter result in fewer events that require crossing processor boundaries, which, in turn, delivers better parallel speedup due to reduced inter-processor communication. On the other hand, well-known hardware artifacts such as caching effects and network dynamics often artificially magnify speedups, such as the few instances of super-linear speedups observed.

6.2. Performance effects due to population unit

To better understand the parallel performance of our discrete event simulation on different population distributions, independent of the complex parameter space spanned by the tuple $\langle R1, R2, D\rangle$, we compare the observed simulation runtimes on two population distributions: one with 1,000 persons/location and 10 locations/region and another with 100 persons/location and 100 locations/region. The total population size of 10,000 persons/region is the same in both cases. For each such comparison, the set of parameters spanned by the tuple $\langle R1, R2, D\rangle$ is kept unchanged. In Figure 17, two population distributions for the same total population size is chosen with [**a a b**]. Similarly, in Figure 18, the population distribution is varied while keeping the settings at [**a c a**].

A close look at the plots reveals a rather curious and counter-intuitive observation. For any given scenario, it can be seen that the performance due to a population distribution with 100 persons/location and 100 locations/region is significantly better than one due to a population distribution with 1,000 persons/location and 10 locations/region. Qualitatively, this translates to the observation that, the finer the population distribution, the better the performance. Conventionally, simulating a less aggregated population distribution (city-level aggregation in a country-wide simulation) is generally expected to take longer than a more aggregated population distribution (state-level aggregation in a country-wide simulation). The observed performance is to the contrary, conveying an exactly opposite runtime behavior.

A more careful examination of this counter-intuitive observation reveals the cause to be that, with increased number of locations per region (that is, less aggregation), the choice of destinations that are local to a processor increases (number of destinations on remote processors decreases). This results in less communication overhead and better runtime. Added to this is also the fact that the time for evaluating the reaction function in Equation (1) grows linearly with the aggregation unit, making finer population units more efficient in runtime. This counter-intuitive effect seems to indicate that it is more favorable to simulate higher fidelity (less aggregate) reaction–diffusion-based

epidemiological models on hierarchical distribution of population. In other words, it is more efficient to simulate the natural decomposition of US population into towns and cities than at more aggregated units of counties or states.

7. Summary and ongoing work

A discrete event formulation, its reverse-computation-based parallel execution, and a performance study of a robust implementation have been presented here for simulating epidemiological outbreaks at high fidelity at the scale of multiple millions of individuals. The focus of this article was more on the computational aspects, as opposed to the use of the simulation for domain science in epidemic simulations. Scalability of the overall system to very large parallel computing platforms has been demonstrated, with the largest being executed on 65,536 processor cores. An important outcome of this work is evidence that PDES technology has now reached a milestone at which very large epidemic simulations can be realized with significant speedup (over $10^4\times$) using large parallel computing platforms.

We are currently incorporating additional modeling concepts, namely, resource constraints and critical thresholds, whose significance was suggested recently.[7] Also being incorporated are prevention, mitigation and intervention mechanisms and their effects. We are also investigating the performance differences between optimistic and conservative execution for the epidemic models. Similar performance comparison of interest is an empirical performance study to compare the performance between copy state saving and incremental state saving in comparison to the reverse computation reported here. In addition, we are pursuing the use of real-life, interconnectivity graphs (e.g. social and transportation networks) for diffusion, along with customization of the system for sponsor agencies and collaborating institutions.

Verification and validation are naturally required before the simulation can be readily employed in actual scenarios. We have performed a verification exercise as reported in Section 5. Validation is a more intensive effort that can be performed using genuine, historical data in terms of reaction and diffusion parameters used in the model. Part of the validity of the reaction equation is obtained from its reported use by Barrett et al.[2] for analyses in government agencies. Since the diffusion network supported in our model can be an arbitrary inter-location mobility network, a validation exercise can be performed by choosing the appropriate probability distributions from US Census Bureau and Bureau of Transportation Statistics for sampling the inter-location travel times.[20] The resultant infection and recovery rates can be whetted by domain (epidemiological) experts. Follow-on work is needed to exercise the entire simulation system by domain experts, to explore effective usage of the scale and speed offered by the system within bonafide solution processes for proactive, reactive and *post facto* operations of epidemic outbreaks.

Acknowledgements

This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. This effort has been partly supported by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), and in part by the DOE Office of Science, Advanced Scientific Computing Research, Career Research Program. This research utilized resources of the National Center for Computational Sciences at ORNL.

References

1. Gojovic MZ, Sander B, Fisman D, Krahn MD and Bauch CT. Modelling mitigation strategies for pandemic (H1N1) 2009. *CMAJ* 2009; 181: 673–680.
2. Barrett CL, Bisset KR, Eubank SG, Feng X and Marathe MV. EpiSimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. Austin, TX: IEEE Press, 2008.
3. Barthelemy M, Barrat A, Pastor-Satorras R and Vespignani A. Velocity and hierarchical spread of epidemic outbreaks in scale-free networks. *Phys Rev Lett* 2004; 92: 178701.
4. Pastor-Satorras R and Vespignani A. Epidemic dynamics in finite size scale-free networks. *Phys Rev E* 2002; 65: 035108.
5. Pastor-Satorras R and Vespignani A. Epidemic spreading in scale-free networks. *Phys Rev Lett* 2001; 86: 3200–3203.
6. Watts DJ and Strogatz SH. Collective dynamics of 'small-world' networks. *Nature* 1998; 393: 440–442.
7. Huang C-Y, Tsai Y-S, Sun C-T, Hsieh J-L and Cheng C-Y. Influences of resource limitations and transmission costs on epidemic simulations and critical thresholds in scale-free networks. *SIMULATION* 2009; 85: 205–219.
8. Epstein J. Modelling to contain pandemics. *Nature* 2009; 460: 687.
9. Bobashev GV, Goedecke DM, Yu F and Epstein J. A hybrid epidemic model: Combining the advantages of agent-based and equation-based approaches. *Proceedings of the Winter Simulation Conference* 2007.
10. Riley S. Large-scale spatial-transmission models of infectious disease. *Science* 2007; 316: 1298–1301.
11. Eubank S, Guclu H and Marathe MV. Modeling disease outbreaks in realistic urban social networks. *Nature* 2004; 429: 180–184.

12. Halloran ME, Ferguson NM, Eubank S, et al. Modeling targeted layered containment of an influenza pandemic in the United States. *Proc Natl Acad Sci USA* 2008; 105: 4639–4644.
13. Germann TC, Kadau K, Longini IM and Macken CA. Mitigation strategies for pandemic influenza in the United States. *Proc Natl Acad Sci U S A* 2006; 103: 5935–5940.
14. Parker J. A flexible, large-scale, distributed agent-based epidemic model. *Winter Simulation Conference*. Piscataway, NJ: IEEE Press, 2007.
15. Linebarger JM, Goldsby ME, Fellig D, Hawley MF, Moore PC and Sa TJ. Smallpox over San Diego: joint real-time federations of distributed simulations and simulation users under a common scenario. In *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*. Piscataway, NJ: IEEE Computer Society, 2007, pp. 7–14.
16. Bauer DW and Mohtashemi M. An application of parallel Monte Carlo modeling for real-time disease surveillance. In *Winter Simulation Conference, 2008 (WSC 2008)*. 2008, pp. 1029–1037.
17. Bisset KR, Chen J, Feng X, Kumar VSA and Marathe MV. EpiFast: A fast algorithm for large scale realistic epidemic simulations on distributed memory systems. *Proc of International Conference of Supercomputing*. 2009, pp. 430–439.
18. Perumalla KS and Seal SK. Reversible parallel discrete event execution of large-scale epidemic outbreak models. In *IEEE/ACM/SCS International Workshop on Principles of Advanced and Distributed Simulation*. Atlanta, GA: IEEE Computer Society, 2010.
19. Carothers C, Perumalla KS and Fujimoto RM. Efficient optimistic parallel simulations using reverse computation. *ACM Trans Model Comput Sim* 1999; 9: 224–253.
20. Perumalla KS and Bhaduri B. On accounting for the interplay of kinetic and non-kinetic aspects in population mobility models. In *European Modeling and Simulation Symposium*. Spain, 2006.

Appendix

A visualization system of the μsik simulator is used to provide animation capabilities for visualizing the run-time dynamics of the simulation, providing highly interactive graphical interface at large event volumes (millions of events). Visualization is enabled for the epidemic simulation to support large-scale scenarios including millions of individuals instantiated in each run. To illustrate the operational and functional aspects of the system as a whole, a few representative snapshots of the animation are shown next.

Figure 19 shows a portion of the event communication pattern among locations (LPs) in a sample simulation of 10 regions with 10 locations per region and 1,000 persons per location (giving a total of 100,000 persons in the scenario). Figure 20 shows a snapshot of the distribution of the persons (dwelling in or in transit) across locations and regions.

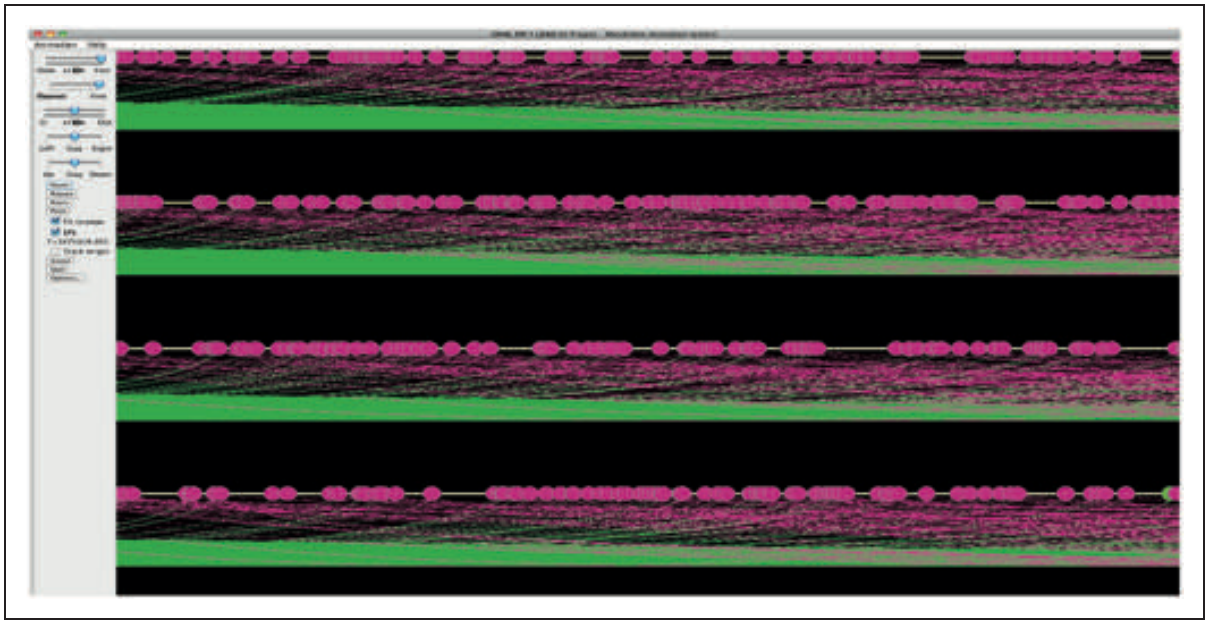

Figure 19. A snapshot of event communication pattern in a sample scenario. Logical processes are displayed along the vertical axis, while their simulation timelines progress horizontally from left to right. Arrival events are drawn in green, departure events in purple. Inter-LP events are not visible at this zoom level.

Figure 20. A set of snapshots in an animation of a simple scenario, shown with color-coded infection states of each person. For simplicity of display, a small scenario is shown, with 35 regions, 20 locations per region, and 100 individuals per location.

Kalyan S Perumallal founded and leads the High Performance Discrete Computing Systems team in the Modeling and Simulation Group at ORNL. He is a Senior R&D Manager in the Computational Sciences and Engineering Division at the Oak Ridge National Laboratory, and an Adjunct Professor at the Georgia Institute of Technology. He earned his PhD in Computer Science in 1999 from the Georgia Institute of Technology. His areas of interest include high performance computing, reversible computing software, parallel discrete event simulation, and parallel combinatorial optimization.

Sudip K Seal is a Research Staff Member in the Computational Sciences and Engineering Division, Oak Ridge National Laboratory. He has a PhD in Theoretical Physics (2002, New Mexico State University) and a PhD in Computer Engineering (2007, Iowa State University, USA). His research interests include parallel algorithms, large scale combinatorial scientific computing, numerical algorithms and parallel discrete event simulations.