Massively Parallel Simulations of Spread of Infectious Diseases over Realistic Social Networks

Abhinav Bhatele[†], Jae-Seung Yeom[†], Nikhil Jain[†], Chris J. Kuhlman[*], Yarden Livnat[‡], Keith R. Bisset[*], Laxmikant V. Kale[§], Madhav V. Marathe[*]

[†]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California 94551 USA
[*]Biocomplexity Institute & Department of Computer Science, Virginia Tech, Blacksburg, Virginia 24061 USA
[‡]Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, Utah 84112 USA
[§]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 USA
E-mail: [†]{bhatele, yeom2, nikhil}@llnl.gov, [*]{ckuhlman, kbisset, mmarathe}@vbi.vt.edu

***Abstract*—Controlling the spread of infectious diseases in large populations is an important societal challenge. Mathematically, the problem is best captured as a certain class of reaction-diffusion processes (referred to as contagion processes) over appropriate synthesized interaction networks. Agent-based models have been successfully used in the recent past to study such contagion processes. We describe EpiSimdemics, a highly scalable, parallel code written in Charm++ that uses agent-based modeling to simulate disease spreads over large, realistic, co-evolving interaction networks. We present a new parallel implementation of EpiSimdemics that achieves unprecedented strong and weak scaling on different architectures — Blue Waters, Cori and Mira. EpiSimdemics achieves five times greater speedup than the second fastest parallel code in this field. This unprecedented scaling is an important step to support the long term vision of real-time epidemic science. Finally, we demonstrate the capabilities of EpiSimdemics by simulating the spread of influenza over a realistic synthetic social contact network spanning the continental United States (~280 million nodes and 5.8 billion social contacts).**

I. Overview of the Problem

An epidemic is an occurrence of cases of illness, specified health behavior, or other health related events clearly in excess of normal expectancy in a community or region. Epidemics caused by infectious diseases continue to take their toll on our society [8]. For example, malaria is said to be the primary cause of between 650,000 and 1.4 million deaths just in 2010.

Traditionally, mathematical and computational modeling of epidemic diffusion has focused on coupled rate equations — differential equation models for completely mixing populations, in which large groups of people interact with each other [6]. Over the years, these models have been successful in providing analytical expressions for statistical epidemic parameters, such as the number of people infected, deaths, etc. But they fail to capture the complexity of human behavior and interactions that serve as a mechanism for disease spread.

An alternative approach uses a combination of network theory and discrete event simulations to study epidemics in *large urban areas* — the main idea is that a better understanding of the characteristics of the social contact network can give better insights into disease dynamics and vaccination/quarantining strategies, which can be used in the epidemic simulation. In this submission, we present a highly scalable, parallel implementation of this alternative approach called EpiSimdemics [11]. In EpiSimdemics, individuals in the population, and each interaction between pairs of them are modeled individually to simulate epidemic diffusion in social contact networks. This is referred to as agent-based modeling.

EpiSimdemics leverages Charm++'s over-decomposition and asynchronous execution to orchestrate a data-dependent message-driven interaction of individuals [7]. High performance is obtained by adaptive overlap of computation and communication, automatic aggregation of fine-grained messages, and use of load balancing. We demonstrate strong and weak scaling of EpiSimdemics using a year 2000 census-based population dataset of continental United States (280 million people, >1 trillion person-to-person interactions over 180 days) to full-scale systems and achieve a simulation rate of 57.8 ms per simulated day on 655,360 cores of Blue Waters. At the highest scale, EpiSimdemics processes more than 100 billion interactions or edges per second.

II. Application Scenario and Challenges

Simulating epidemic diffusion over realistic social networks in parallel is challenging for various reasons. First, the size and scale of these systems is extremely large (e.g., pandemic planning at a global scale requires models with 6 billion agents). Second, the networks are highly unstructured and the computations involve complicated dependencies, leading to high communication cost and making standard techniques of load balancing and synchronization ineffective. Third, individuals are not identical — this implies that models of individual behavioral representation cannot be identical. And most importantly, epidemics, the underlying interaction network, public policies and individual agent behaviors co-evolve, making it nearly impossible to apply standard model reduction techniques that are successfully used to study physical systems.

In the EpiSimdemics model, a person interacts with another person when they both visit a location at the same time. The longer the interaction period, the higher the chance of transmitting a disease if one person is infectious and the other person is susceptible. This representation of *interactions* between people as *visits* to locations avoids explicit messages

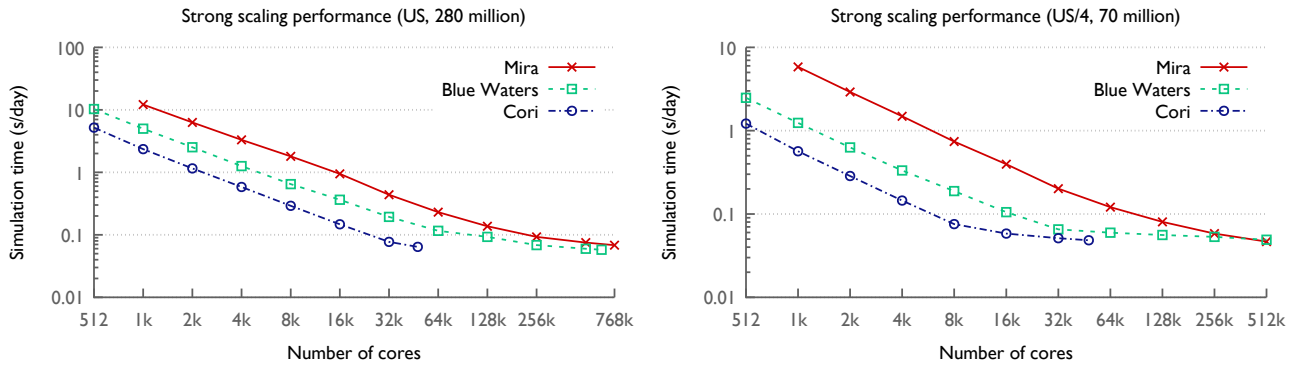

Fig. 1. Strong scaling performance of EpiSimdemics on three platforms using two datasets. High efficiency is obtained on all machines to large core counts.

between every pair of interacting persons. We rely on a bipartite graph between people and locations to represent the social contact network and explicitly represent each individual person and location in the input dataset. The main computation involves processing visits, which is not expensive. As a result, a large fraction of the time is spent in communication, unlike physical simulation codes that are heavy on floating point operations. Hence, EpiSimdemics is closer to parallel graph analytics codes that spend most of their time in memory accesses, integer operations and branching. All of these make the problem extremely challenging to parallelize and scale out.

III. Performance Metrics and Results

We use two metrics to evaluate and compare the scalability of the code: time required to simulate one day (s/day) and number of people-people interactions processed per second (interactions/s). The first metric is calculated based on the total time, T reported by the timer, as

$$\text{Time per day, } T_d = T/d,$$

In this study, we use d=180 days for all the simulations.

The second metric, R_I, the rate of processing interactions is calculated by obtaining the total number of person-person social contacts, N_I (also referred to as interactions or edges) reported by the code (summed over 180 days) and dividing it by the total time T,

$$R_I = N_I/T$$

This metric is similar to the performance metric used by the Graph500 community — traversed edges per second (TEPS). It should be noted that it would be unfair to compare the TEPS reported for EpiSimdemics simulating real-world problems with those of Graph500 benchmarks.

A. Experimental Setup

We use a realistic social contact network spanning the continental United States, synthesized using census and relevant data [1]. This network has 280.4 million agents or humans and 5.8 billion person-person social interactions or edges. We construct a second dataset (US/4) which is one-fourth the size of the full US dataset. The US/4 dataset includes 70.8 million people of the states in the West Census region (Mountain and Pacific divisions) plus Oklahoma, Kansas and Nebraska. All runs simulate 180 days or 6 months of disease spread.

We conducted our experiments on three Cray and IBM supercomputers: Blue Waters, a Cray XE6/XK7 at the National Center for Supercomputing Applications (NCSA); Cori, a Cray XC30 at The National Energy Research Scientific Computing Center (NERSC); and Mira, an IBM Blue Gene/Q at the Argonne Leadership Computing Facility (ALCF). We used Charm++ version 6.7.0 for building EpiSimdemics.

B. Performance Results

Figure 1 presents scaling results of EpiSimdemics simulating the US and US/4 datasets on the three systems described earlier. On each system, the simulation is performed on the smallest core count on which the dataset fits in memory and up to the largest number of cores available on the system. The x-axis in these plots shows the number of cores and the y-axis shows the time spent in simulating one day. On all machines and for both datasets, high efficiency is obtained up to very large core counts.

On Cori, for the US dataset, a speed up of 67.5× is obtained with a 64× increase in core count from 512 to 32,768 cores, i.e., we obtain super-linear speed up as we scale to two-thirds of the full system. This perfect scaling is a result of highly accurate load balancing and increased availability of network bandwidth as we run on more nodes. Scaling to the full Cori system (49,152 cores) further reduces the execution time and provides the best performance of 11.5 seconds to simulate 180 days (64.15 ms/day). This level of **performance is unprecedented** for simulating the entire US population. Even if we include the time for startup and file input/output, we can complete the entire simulation in a few minutes. Similar good performance is achieved for the US/4 dataset, where perfect scaling is observed till 4,096 cores and the best execution time of 8.29 seconds is observed on 49,152 cores for simulating 180 days (46.06 ms/day).

On Mira and Blue Waters, which have lesser memory and network bandwidth per core in comparison to Cori, execution is comparatively slower on lower core counts. However, the code scales with **good parallel efficiency to hundreds of**

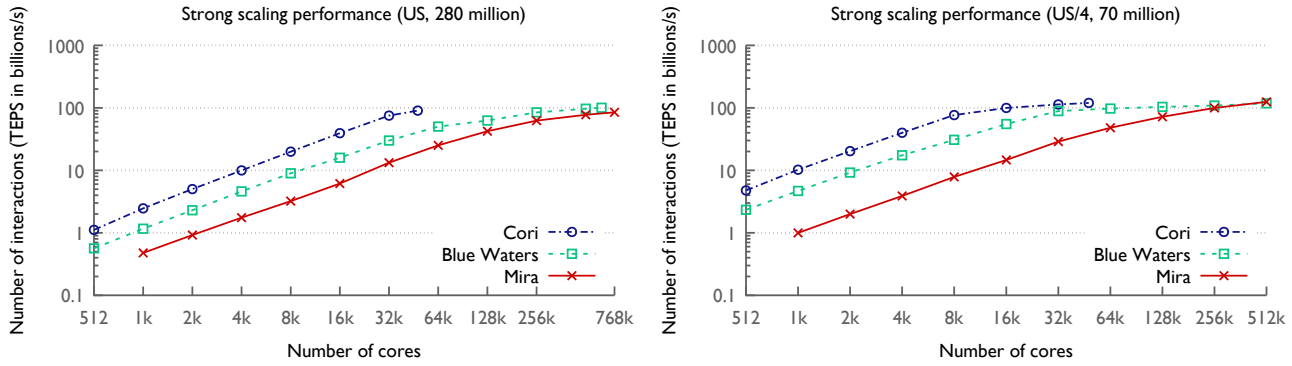

Fig. 2. The processing of person-person edges or interactions, measured in terms of traversed edges per second (TEPS) is a key aspect of epidemic simulations. On all systems, as the core count is increased, ∼100 billion interactions are processed per second (average over 180 simulated days).

thousands of cores. At 1,024 cores, EpiSimdemics takes 423.62 seconds on Cori, 898.44 seconds (2.1×) on Blue Waters, and 2188.08 seconds (5.2×) on Mira to simulate 180 days of the US dataset. When using all cores of Blue Waters (655,360 cores) and Mira (786,432 cores), the code takes only 10.4 and 12.3 seconds respectively to simulate 180 days of the US dataset.

For the US/4 dataset, the time to simulate 180 days is 8.84 and 8.38 seconds on Blue Waters and Mira respectively, at the largest core counts where we ran this dataset. It is to be noted that strong scaling on very large counts of Blue Waters and Mira is limited by the heavy-tailed degree and load distribution of these datasets (Figure 3).

The processing of person-person edges or interactions is a key aspect of epidemic simulations. Figure 2 shows that EpiSimdemics provides good scalability when executed on different systems. On 1,024 cores of Cori, EpiSimdemics processes 2.4 billion interactions per second, which implies processing the interaction of every person with ∼9 other people per second. In contrast, at the largest core counts, it processes ∼100 billion interactions per second (TEPS). This implies that, on an average, EpiSimdemics can simulate the interaction of every person in the United States with ∼350 other people in a second when the simulation is done on the full system (Cori, Blue Waters, or Mira). This is a remarkable achievement to support real-time epidemic science studies.

IV. The Solution and Innovations

We now describe the parallel implementation of agent-based modeling in EpiSimdemics to simulate epidemic diffusion, and the optimizations to improve scaling and efficiency.

A. Overview of the Algorithm

EpiSimdemics uses a hybrid time-stepped and discrete-event simulation approach. The time step is typically one simulated day and within each time step, a discrete event simulation is performed. There are two major computation phases in each time step (`while` loop over number of simulated days) as shown in Algorithm 1. In the first phase (lines 2–8), each person, p, identifies locations to visit depending on the visit schedule and the person's health state, h_p. Then, for each such visit, it sends a message m to the destination location l. In the second phase (lines 9–29), each location l processes received visit messages by first converting them into events, i.e., arrival and departure on a particular sublocation, l_s (lines 10-14), and then by executing a sequential discrete event simulation (lines 15–28) to identify interactions and to compute disease transmission (lines 21–26).

Algorithm 1: Simulating Epidemic Diffusion

```
1  while d ≤ d_max do
2      for p ∈ P do
3          Evaluate scenario trigger conditions;
4          Update health state h_p, if necessary, and reevaluate triggers;
5          foreach v ∈ V_p ( visit schedule of p) do
6              Send visit message m to location l;
7          end
8      end
9      for l ∈ L do
10         foreach m destined for l do
11             Determine the sublocation l_s to visit;
12             Create an arrival and departure event for each visit;
13             Put the events into the event queue q_e of l;
14         end
15         Reorder q_e by the time of event in ascending order;
16         foreach e ∈ q_e do
17             if e is arrival then
18                 Put p into sublocation l_s;
19             else
20                 Remove p from sublocation l_s;
21                 foreach p' currently in l_s do
22                     Compute disease transmission probability q
                       between p' and p;
23                     if q > threshold then
24                         Send infection message to the infected
                           person (p or p');
25                     end
26                 end
27             end
28         end
29     end
30     d++;
31 end
```

In the parallel implementation of Algorithm 1, all the people and locations in the input population data are distributed among all processes. Each phase generates fine-grained communication during the time step and results in

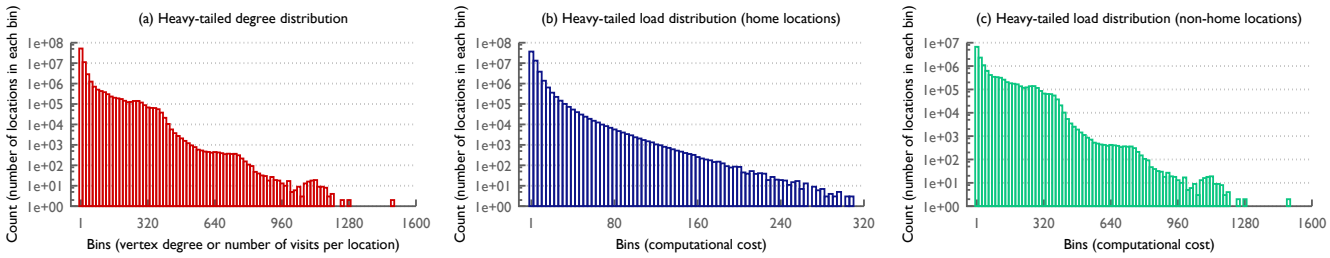

Fig. 3. Histograms showing the heavy-tailed distribution of the input data w.r.t. (a) vertex degree (number of visits per location), (b) estimated computational cost of home locations, (c) estimated computational cost of non-home locations (such as schools, shops etc.)

visit and *infection* messages being sent to remote locations and people respectively. Since the number of visit and infection messages is not known apriori, we use quiescence detection provided by Charm++ to ensure that all messages have indeed been received. Computation in each phase can be performed in parallel on all processes and is scheduled adaptively by Charm++ on message arrival.

B. Performance Optimizations

The initial distribution of people and locations to processes should aim to optimize data locality and balance the amount of computation and communication among all processes. Optimizing data locality requires the placement of interacting people and locations on the same or "nearby" processes (on the same compute node). Balancing the computational and communication load among processes requires estimating the work and communication associated with each person and location. Both of these tasks are challenging due to the heavy-tailed distribution of the population data w.r.t the location vertex degree (number of people visiting each location, Figure 3(a)) and w.r.t. the computational cost of the locations (Figures 3(b) and (c)). Some locations are visited by a significantly large number of people, increasing their computational and communication cost. Their large cost also introduces challenges for load balancing due to a large minimum grain size.

Locality-aware Data Distribution: The default data distribution in EpiSimdemics uses a round-robin scheme, which is good for load balancing without requiring any information about the data or the platform. This scheme has data locality issues because interacting people and locations might end up on different processes. In order to optimize data locality, we have used graph partitioning to partition the bi-partite graph of people and locations in the past [11]. However, due to time constraints, this partitioning was done offline and required partitioning the input data for each node count that we wish to run on. In order to eliminate these restrictions, we have developed a fast online scheme that exploits the ZIP code of locations to co-locate geographically nearby locations on the same or nearby processes.

This ZIP code based data distribution is performed at runtime after the input data is read and does not require any offline data processing. The idea is simple — all locations are sorted by their ZIP code with the assumption that nearby ZIP codes represent areas that are close on the map. This is akin to using space filling curves for linearizing and distributing data in three-dimensional physical simulation codes while maintaining data locality. After the initial sorting by ZIP codes, locations are divided among the processes based on an estimated cost to achieve computational load balance. Figure 4 compares the performance improvements from using the new ZIP code based data distribution and the default round-robin scheme. On Blue Waters, the new scheme leads to performance improvements between 35 and 44% at different core counts.

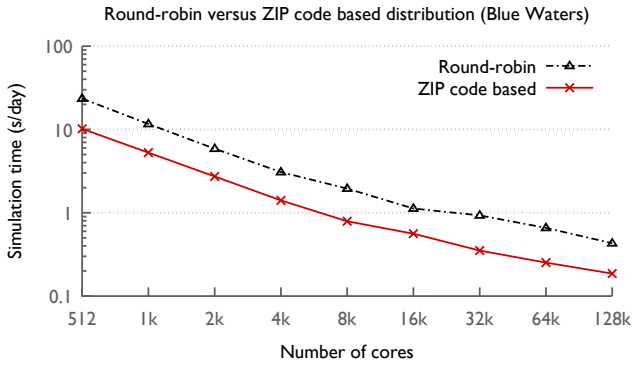

Fig. 4. Performance impact of using different data distribution schemes

Cost Model for Load Balancing: The number of locations assigned to each process depends on an estimated cost of the computation and communication associated with each location. In [11], we used an architecture-specific cost model that is a function of the number of visits and interactions to each location. When we started running on multiple architectures, we realized that the number of visits is a good indicator of the architecture-independent cost of the work or load associated with each location. Further analysis showed that the cost of home locations is better correlated with the number of people attached to each home location than the number of visits. Home locations refer to the locations that are homes as opposed to non-home locations such as schools, shops, etc. Hence, in this work, we use a hybrid cost model in which the estimated cost of a home location is the number of associated people and that of a non-home location is the number of visits. Figure 5 presents the performance impact of using different cost estimates for balancing load. At 262,144 cores on Mira

(Blue Gene/Q), the performance based on the hybrid cost is better than that from the visit cost by nearly 2.5×.

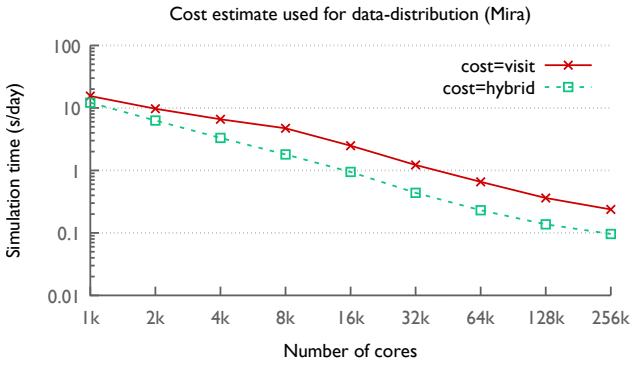

Fig. 5. Performance impact of cost estimate used for load balancing

Communication Optimizations: The first phase of the simulation sends many small *visit* messages. In order to minimize the overhead of sending small messages and to utilize the network better, we use the general purpose TRAM library in Charm++ to aggregate messages. TRAM combines small messages and routes the aggregated messages on a virtual mesh topology. This optimization leads to significant performance improvements at high core counts. Another optimization to facilitate overlap of communication and computation involves alternating the creation and sending of batches of visit messages on each process. This ensures that we do not saturate the network with messages in bursts.

V. Impact of the Solution

The excellent efficiency and extreme scaling of EpiSimdemics has significant implications for the epidemic studies that can be performed using it. Real time planning and response by governments during an epidemic outbreak often require simulation turn around times of less than 24 hours. We have provided this type of support to federal agencies during H1N1 and Ebola outbreaks. It is critical to not only have the capabilities to assess the effects of different interventions on national and international populations, but it is also important to perform these computations quickly, so that multiple scenarios can be evaluated. Rapid response also enables the team to incorporate new information in to the simulation as it becomes available, which sometimes happens on a daily basis.

Here we present a representative study of the impact of vaccination on controlling an epidemic outbreak. All simulations are seeded stochastically, but all use the following process. In each of the first four days of a simulation, agents in two or three U.S. cities are exogenously infected with a probability of 5×10^{-6}. These cities were chosen because they host the busiest airports in the country and thus represent a wide range of initiation sites that might arise from incipient virus spreading from air travel. From day 5 onward, all infections result only from susceptible agents contracting the virus from infectious ones while they are co-located.

Interventions are implemented as follows. When 1% of the population (about 2.8 million people) is infected, the intervention is triggered. Each person still in the susceptible state is vaccinated with probability, p. We use $p = 0.10$, 0.25, and 0.50, in addition to the base case of no intervention ($p = 0.0$). Each person that is vaccinated has full immunity, i.e., he or she cannot contract the virus and hence also cannot transmit it to others.

Figure 6 provides the simulation results — the number of currently infectious people as a function of time, with error bars denoting ±1 standard deviation over 20 repetitions of each scenario. Vaccinating 10% (i.e., $p = 0.10$) of the remaining susceptible population reduces the peak number of currently infectious people by about 23% at day 63. Vaccinating 50% of the remaining susceptible population reduces the peak number of currently infectious people by about 80%. In the latter case, there is a sufficient number of vaccinations to decelerate the epidemic almost immediately. Figure 7 shows the disease spread for $p = 0.10$ on simulated day 56, which is the day when the rate of infection peaks for the base case of no intervention.

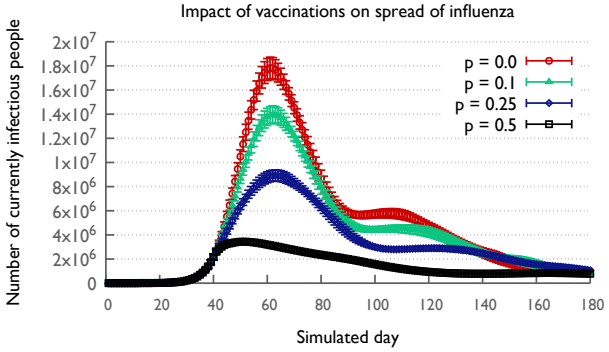

Fig. 6. Simulation results of a vaccination study performed using EpiSimdemics. The plot shows the number of infectious people per day as the simulation progresses for different vaccination scenarios.

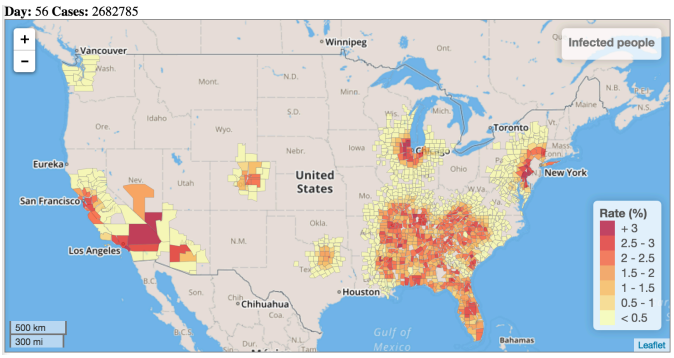

Fig. 7. Simulation results of a vaccination study performed using EpiSimdemics. The visualization shows the geographical spread of the infection on day 56 for $p = 0.10$.

Being able to run one repetition of an intervention scenario described above in about 6 minutes on 16,384 cores of Blue Waters is a key element of this process. In the study described

TABLE I
REPORTED PERFORMANCE OF STATE-OF-THE-ART SIMULATORS AND COMPARISON WITH THIS WORK

Code	No. of agents	Simulated period	Machine	No. of cores	Simulation time
FRED [5]	289 million	Unknown	Blacklight @ PSC	16 threads	4 hours
GSAM [9]	262.9 million	Unknown	2.4 GHz Opteron 8216	4 cores	72 minutes
SPaSM [4]	281 million	180 days	2.4 GHz Intel Xeon	256 cores	8–12 hours
EpiSimdemics [11]	280.4 million	120 days	Blue Waters @ NCSA	16,384 cores	134.89 s
This work	280.4 million	180 days	Blue Waters @ NCSA	655,360 cores	10.41 s

here, we performed 20 repetitions of each vaccination scenario (p = 0.0, 0.10, 0.25, and 0.50), resulting in 80 simulations in total. If we use all 655,360 cores of Blue Waters, we can set up an ensemble run which simulates 40 simulations in parallel on 16,384 cores each. This would allow us to complete this entire study in 12 minutes using most of the Blue Waters cores. This suggests that we can simulate, for instance, 20 different intervention scenarios in an hour and 240 scenarios with 5 parameters (each of which can take 3 distinct values) in half a day. This kind of epidemic simulation capability is **unprecedented and can be extremely valuable** to epidemiologists and governments in preventing epidemic outbreaks. To the best of our knowledge, no other simulation tool has this level of performance and turn around times.

VI. COMPARISON TO RELATED APPROACHES

It is difficult to compare the performance of large, agent-based, epidemic simulation systems because most publications in the field do not include detailed high performance computing data (Table I). The Framework for Replication of Epidemiological Dynamics (FRED) [5] uses OpenMP for shared memory concurrency. On the supercomputer Blacklight (an SGI Altix UV 1000) at the Pittsburgh Supercomputer Center, a simulated pandemic over the entire U.S. population takes approximately four hours using 16 threads.

A Java-based, distributed agent-based epidemic simulator called the Global-Scale Agent Model (GSAM) is described in [9]. It is designed to simulate a world-wide epidemic with some 6 billion agents. A simulation of 262.9 million agents, using two dual-core 2.4 GHz Opterons completes in 4321 seconds and achieves 0.734 million bps. Germann et al. [4] performed agent-based simulations on a 281 million synthetic population of the U.S. They used the Scalable Parallel Short-range Molecular dynamics code (SPaSM), which is a C and MPI-based simulator. They quote a typical production run as a single simulation on 256 2.4 GHz compute cores of an Intel Xeon-Myrinet cluster. On this system, a representative run took 8 to 12 hours to simulate 180 days.

Perumalla and Seal [10] have performed distributed epidemic simulations on multiple supercomputers. Their epidemic model is based directly on prior EpiSimdemics work [2]. Their simulation engine employs an optimistic discrete-event approach that includes reverse computations. On an 8,192-core Blue Gene/P system, they achieved a speedup of 5,500 on 8,192 cores. With a Cray XT5 system, using 65,536 cores, they attained a speedup of over 10,000.

Prior works on EpiSimdemics [2], [11] in addition to EpiSim, which simulated epidemics in Portland, Oregon [3], are most closely related to the current work. Marked improvements in performance and scalability of EpiSimdemics were achieved with a re-implementation that uses the Charm++ runtime system [7]. This version surpassed the speedup attained in Perumalla and Seal [10] by $>5\times$ by achieving a speedup of 58,649 on 360,448 cores (16% efficiency) on a Cray XE6 [11]. The current version obtains a speedup of 182,073 (23% efficiency) on 786,432 cores of Blue Gene/Q.

ACKNOWLEDGMENT

This work was performed under the auspices of the U.S. DOE by LLNL under Contract DE-AC52-07NA27344 (LLNL-CONF-690723), and was supported in part by DTRA (CNIMS HDTRA1-11-D-0016-0001, HDTRA1-11-1-0016), and NSF (NetSE CNS-1011769, SDCI OCI-1032677, MIDAS 5U01GM070694-11, DIBBs ACI-1443054, Big Data IIS-1633028). The authors thank Bill Kramer, Greg Bauer, Sharif Islam, Richard Gerber, Woo-Sun Yang, Christopher Knight, Scott Futral, and Greg Tomaschke for help with allocations on Blue Waters, supported by NSF (OCI-0725070 & ACI-1238993) and the state of Illinois; and at the NERSC Center and ALCF, DOE Office of Science User Facilities supported by the U.S. DOE under Contracts DE-AC02-05CH11231 and DE-AC02-06CH11357 respectively.

REFERENCES

[1] C. Barrett, R. Beckman, M. Khan, V. A. Kumar, et al. Generation and analysis of large synthetic social contact networks. In *WSC*, 2009.

[2] C. Barrett, K. Bisset, S. Eubank, X. Feng, and M. Marathe. EpiSimdemics: an Efficient Algorithm for Simulating the Spread of Infectious Disease over Large Realistic Social Networks. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008.

[3] S. Eubank, H. Guclu, V. S. A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429:180–184, 2004.

[4] T. C. Germann, K. Kadau, I. M. Longini, and C. A. Macken. Mitigation strategies for pandemic influenza in the united states. *Proc. of the National Academy of Sciences*, 103(15):5935–5940, 2006.

[5] J. J. Grefenstette et al. Fred (a framework for reconstructing epidemic dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations. *BMC Public Health*, 13(1):1–14, 2013.

[6] H. W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42:599–653, 2000.

[7] L. V. Kale and A. Bhatele, editors. *Parallel Science and Engineering Applications: The Charm++ Approach*. CRC Press, Oct. 2013.

[8] N. M. Molinari, I. R. Ortega-Sanchez, M. L. Messonnier, W. W. Thompson, P. M. Wortley, E. Weintraub, and C. B. Bridges. The annual impact of seasonal influenza in the US: Measuring disease burden and costs. *Vaccine*, 25(27):5086–5096, 2007.

[9] J. Parker and J. M. Epstein. A distributed platform for global-scale agent-based models of disease transmission. *ACM Trans. Model. Comput. Simul.*, 22(1):2:1–2:25, Dec. 2011.

[10] K. S. Perumalla and S. K. Seal. Discrete event modeling and massively parallel execution of epidemic outbreak phenomena. *Simulation*, 88(7):768–783, July 2012.

[11] J. Yeom, A. Bhatele, K. R. Bisset, E. Bohm, A. Gupta, L. V. Kale, M. Marathe, D. S. Nikolopoulos, M. Schulz, and L. Wesolowski. Overcoming the scalability challenges of epidemic simulations on Blue Waters. In *28th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2014.