Overcoming the Scalability Challenges of Epidemic Simulations on Blue Waters

Jae-Seung Yeom[1,2], Abhinav Bhatele[3], Keith Bisset[2], Eric Bohm[4], Abhishek Gupta[4], Laxmikant V. Kale[4], Madhav Marathe[1,2], Dimitrios S. Nikolopoulos[5], Martin Schulz[3], Lukasz Wesolowski[4]

[1]*Department of Computer Science, Virginia Tech, Blacksburg, VA 24061 USA*
[2]*Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA 24061 USA*
[3]*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551 USA*
[4]*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA*
[5]*School of EEECS, Queen's University of Belfast, Belfast, Northern Ireland BT9 5BN UK*
Email: [2]{*jyeom, kbisset, mmarathe*}*@vbi.vt.edu,* [3]{*bhatele, schulzm*}*@llnl.gov*
[4]{*ebohm, gupta59, kale, wesolwsk*}*@illinois.edu,* [5]*d.nikolopoulos@qub.ac.uk*

***Abstract*—Modeling dynamical systems represents an important application class covering a wide range of disciplines including but not limited to biology, chemistry, finance, national security, and health care. Such applications typically involve large-scale, irregular graph processing, which makes them difficult to scale due to the evolutionary nature of their workload, irregular communication and load imbalance. EpiSimdemics is such an application simulating epidemic diffusion in extremely large and realistic social contact networks. It implements a graph-based system that captures dynamics among co-evolving entities. This paper presents an implementation of EpiSimdemics in Charm++ that enables future research by social, biological and computational scientists at unprecedented data and system scales. We present new methods for application-specific processing of graph data and demonstrate the effectiveness of these methods on a Cray XE6, specifically NCSA's Blue Waters system.**

***Keywords*-contagion simulations; scalability; performance; graph processing; social contact networks**

I. Introduction

With an increasingly urbanized and mobile population, the likelihood of a worldwide pandemic is increasing. To understand and combat such events, scientists require accurate simulations that help them study how contagions spread among the population of a state, country or, ultimately, the entire planet. However, with rising input sizes and accuracy requirements coupled with strict deadlines for simulation results, e.g., for real-time planning during the outbreak of an epidemic, simple computational approaches are no longer sufficient. For example, the analysis necessary during the 2009 outbreak of the avian flu (H1N1) required simulation turnaround times of under 24 hours for several weeks. Therefore, we must expand the use of high performance computing (HPC) approaches and, in particular, push the boundaries of scalability for this application area.

EpiSimdemics is an agent-based simulation framework for contagion simulation that can be used to model a wide range of epidemic scenarios, as well as the impact of counter measures [1]. It has been used in multiple sponsor requested studies and for determining potential outcomes during the early days of the H1N1 pandemic [2]. This integration into the 24-hour decision cycle of the federal government's response to such a crisis would not have been possible without the development of highly optimized modeling software. The analysts performed course-of-action analyses to estimate the impact of closing schools and shutting down workplaces.

EpiSimdemics is implemented using the Charm++ programming model [3] and has shown good scalability up to modest cluster sizes of several hundred to a thousand cores. When scaling to a larger number of cores, however, EpiSimdemics faced severe scalability bottlenecks that prohibited its use for large, time-sensitive problems.

In this paper, we analyze the source of these scalability limitations and discuss a set of novel optimizations to overcome them. We first focus on the impact of using graph data with heavy-tailed degree distributions, which is common in social network graphs. We show how such a property limits scaling and how we transform graphs to achieve a better load balance via graph partitioning.

We make the following specific contributions in this paper:

- We analyze the challenges in scaling a state-of-the-art contagion simulation code, EpiSimdemics, and connect them to the heavy-tailed properties of the input graph.
- We introduce a workload model that allows state-of-the-art graph partitioners to use custom, application-specific load balancing constraints for EpiSimdemics.
- We propose a technique to preprocess the input graph to split heavy nodes, which enables graph partitioners to produce a more balanced workload distribution.
- We implement and evaluate a series of communication optimizations for irregular graph-processing applications, including message aggregation.
- We demonstrate unprecedented strong scaling of a contagion simulation on over 352K cores of Blue Waters at National Center for Supercomputing Applications (NCSA), one of the largest HPC systems in the world.

The fastest known simulation [4] reported a speedup of

10,000 on 64K cores (15.2% efficiency) on Cray XT5. Our approach achieves a speedup of 14,357 (22% efficiency) on the same number of cores on Cray XE6. Further, we demonstrate that our implementation of EPISIMDEMICS can scale up to 360,448 cores and achieve a speedup 58,649 (16.3% efficiency), more than a five-fold increase in the number of cores with slightly improved efficiency.

II. EPISIMDEMICS

EPISIMDEMICS is a contagion diffusion simulation code, which relies on several underlying base technologies. Here, we describe the simulation algorithm and its implementation.

A. Contagion Simulation Structure

EPISIMDEMICS is an agent-based simulator that models the spread of contagious disease over social contact networks. It is based on a hybrid time-stepped, discrete-event simulation (DES) approach. Its input is a bipartite graph consisting of person and location nodes, with edges between them representing a visit by a person to a specific location at a specific time. This graph is a synthetic network based on census and other data [5]. We call this the person-location graph. It is a compact representation, with an average degree of 5.5 for person nodes and 21.5 for location nodes.

Table I lists the population sizes of some representative US states. Note that the graph can evolve over time as people's health state changes and interventions such as school closures are applied. The person-location graph is used to implicitly construct a person-person graph, whose edges represent the colocation of two people in time and space and which is ultimately used to determine any disease transmission between colocated people.

Table I
POPULATION DATA OF VARIOUS SIZES BASED ON A 2009 AMERICAN COMMUNITY SURVEY. US INCLUDES 48 CONTIGUOUS STATES AND DC.

	data name	visits	people	locations
US	(United States)	1,541,367,574	280,397,680	71,705,723
CA	(California)	183,858,275	33,588,339	7,178,611
NY	(New York)	98,350,857	17,910,467	4,719,921
MI	(Michigan)	52,534,554	9,541,140	2,490,068
NC	(North Carolina)	47,130,620	8,541,564	2,289,167
IA	(Iowa)	15,280,731	2,766,716	748,239
AR	(Arkansas)	14,803,256	2,685,280	739,507
WY	(Wyoming)	2,756,411	499,514	144,369

A person's health state is tracked using a probabilistic timed transition system (PTTS), a finite state machine with the addition of a dwell time (the time a person will remain in a state before automatically transitioning to the next state) distribution for each state, and sets of probabilistic transitions between states. Different sets of transitions are used, depending on the treatment received by the person, such as vaccination. EPISIMDEMICS has a domain-specific language for specifying complex interventions and behavior, such as vaccinations, school closures, and anxiety levels [6].

B. The EpiSimdemics Core Algorithm

We take advantage of the fact that most infectious diseases have a latent period, the time between a person becoming infected and being able to infect others. This lets us process all of the interactions for a time step in parallel, without affecting causality. The basic algorithm for each time step (currently one simulation day) is:

1) In parallel, each person recalculates their health state and decides on the locations to visit, based on their current normative schedule, health state, and public policy such as school closings. For each location visited, the object representing the person sends a "visit" message to the object representing the visited location with the ID of the person, the start time and the end time of the visit, as well as the person's health state.
2) Synchronization to ensure all visit messages have been received, as receivers have no prior knowledge of how many messages to expect and from whom.
3) In parallel, each location constructs a sequential and local DES by converting each visit message into an arrive event and depart event. The DES is executed, computing the interactions between each pair of susceptible and infectious people who are at the location at the same time. For each interaction that results in disease transmission, an "infect" message is sent to the infected person.
4) Synchronization to ensure receipt of all infect messages.
5) In parallel, each person that received an infect message updates its health state.
6) Global system state is updated (e.g., number of currently infected people).

While this design requires two global synchronization points for each iteration and therefore leads to a bulk-synchronous model, EPISIMDEMICS typically only requires the execution of a moderate number (120–180) of fairly long simulation iterations on most inputs, representing three to four months of simulated time. This helps mask the effects of the bulk synchronous model on scalability.

C. Message-driven Design

EPISIMDEMICS is implemented in a parallel language called CHARM++ [3], which is a C++-based parallel programming model accompanied by a message-driven asynchronous runtime. The underlying idea is to over-decompose the computation in the application into smaller units called *chares*, i.e., into significantly more units than available physical processors, and to let the runtime then assign a set of work units to each physical processor, which enables a fine grained load balancing. Chares can either be data units, work units, or both. However, implementations must choose the right granularity of splitting work into chares to find the right tradeoff. A large number of chares, each with little work increases flexibility, but also results in higher overhead, while a small number of larger chares minimizes overhead but limits the ability to exploit over-decomposition.

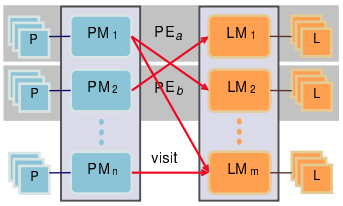


Figure 1. EPISIMDEMICS implemented in CHARM++

We follow a two-level hierarchical data distribution technique to find the right tradeoff, as shown in Figure 1. At the first level, we create two types of chares, LocationManagers (LM) and PersonManagers (PM), each able to manage multiple second level objects representing individual locations and persons, respectively. We then distribute the person and location objects among the elements of the corresponding *chare arrays* (LM and PM). The individual chares in both arrays handle the computation and communication of all location or person objects assigned to them. The CHARM++ runtime then maps the chare arrays (representing LMs and PMs) to processes. As a consequence, different object to manager and manager to processor mappings can result in different communication patterns with potentially varying degrees of efficiency.

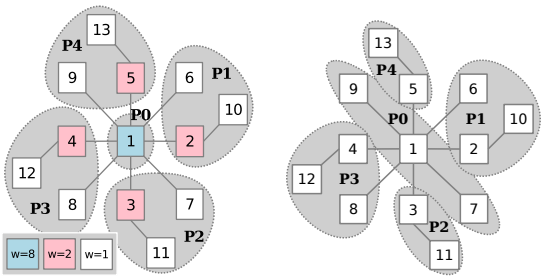
III. SCALABILITY CHALLENGES WITH SOCIAL NETWORK DATA

The input graphs to EPISIMDEMICS, which are derived from real-world data [5], typically follow heavy-tailed degree distributions. This is common for social network graphs [7]. This property has a profound impact on how to handle and scale such data sets, as we show in the rest of the paper.

Heavy-tailed load distributions make load balancing difficult when partitioning. Furthermore, partitioning to optimize locality of data by minimizing total edge cuts may often only be achieved by sacrificing load balancing. Figure 2(a) and (b) show the optimal partitioning in terms of load balancing without considering edge cuts, and edge cuts without considering load balancing, respectively. In this example, the most heavily-loaded node (Node 1 with weight 8) has the most edges and balancing load requires cutting all edges around this node in Figure 2(a). Partitioning focusing on edge cuts, on the other hand, would lead to the distribution in Figure 2(b) with an edge cut of 6 vs. 8 in Figure 2(a). However, the ratio of the maximum load per partition to the average is 1.67 in Figure 2(a) and 2.08 in Figure 2(b), showing the advantage of the distribution in Figure 2(a). Section III-B further discusses this problem and Section III-C describes how we address the problem in EPISIMDEMICS using an application-specific decomposition strategy.

A. A Model to Estimate Work Load

Load distribution is important to achieve high scalability and performance. Existing graph partitioners, such as METIS [8], allow users to specify the load balance constraint



(a) total 8 cuts, max load per partition = 8

(b) total 6 cuts, max load per partition = 10

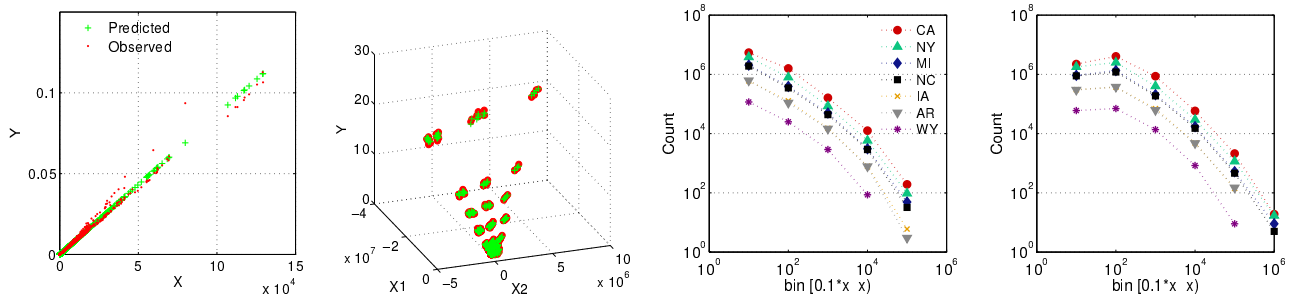
Figure 2. The optimal 5-way partitioning result for minimizing (a) load imbalance and (b) edge cut. Node 1 has weight 8, and Nodes 7 and 9 have weight 1 each.

in terms of the tolerable variance in the sum of vertex weights per partition. This requires a model to estimate the workload in our graphs so that we can assign a weight to each work unit (vertex) before passing graphs to the partitioner. We use METIS for our investigation as it supports multi-constrained partitioning [8], allowing us to assign a vector of weights to each vertex. Each element of the vector is associated with a unique load balancing constraint for a specific phase of the computation. We adopt this mechanism for partitioning population data sets in bipartite graphs representing the dual-phase computation discussed in Section II.

We observe that the amount of computation per person is roughly proportional to the number of messages that each person generates, which shows no significant variance (avg= 5.5, σ=2.6 for the US population data). Thus, we approximate the load of a person vertex as the number of messages the person generates. On the other hand, the computation per location varies significantly and requires a more detailed estimation. For this, we adopt a function approximation approach rather than analytical modeling. We define load as the relative processing time. We consider three application state variables (number of arrival-departure events, sum of interactions and sum of the reciprocal of interactions) for the model input, rather than machine parameters. The latter two are only available at run time.

Note that we distinguish between static load and dynamic load. In EPISIMDEMICS, the amount of computation is not deterministic. Two of the major sources of non-determinism are health state changes and interventions, described in Section II-A. We do not attempt to address the dynamic load variation by static load balancing. Rather, we focus on the statically predictable portion of the workload by using a priori information such as the number of events.

We build a model that maps events to the static load of the location as shown in Figure 3(a). Figure 3(c) shows the distribution of in-degree per location which is the number of unique visitors strongly correlated to the number of events. Figure 3(d) shows the distribution of the load per location estimated by the model. We use a piecewise linear regression to approximate the non-linear dependence that exists between



(a) Static load model (b) Dynamic load model (c) Degree distribution (d) Static load distribution

Figure 3. The load estimation model estimates the load of each location work unit on Blue Waters based on the node degree.

the location computational load and events as follows:

$$\begin{aligned} X' &= \mu \cdot X \\ Y_a &= 6.09\times10^{-6}+7.72\times10^{-7}X' \\ Y_b &= -1.25\times10^{-4}+8.67\times10^{-7}X' \\ Y &= Y_a \cdot S(\varphi - X') + Y_b \cdot S(X' - \varphi) \end{aligned}$$

where X is the number of events, Y is the load and $S(t) = 1/(1+\rho \cdot e^{-t})$. φ is the cross over point between the two linear models and determined experimentally. ρ is set to adjust the smoothness of the transition from one model to another. In practice, we build the model by measuring LocationManagers' processing time due to the limited timer precision but apply it to a location by scaling the input parameter with μ. Figure 3(a) validates our model against runs on Blue Waters, where we observe 5% error on average. The model shown in Figure 3(b) relies on the aforementioned on-line information to estimate dynamic load. Thus, it is not used for graph partitioning.

B. Applying Graph Partitioning to Social Network Graphs

Originally, we assign objects to Charm++ chares round-robin (RR) to approximate static load balancing. However, this is not optimal in terms of load balance and data locality. EpiSimdemics also supports an interface to apply external partitioning methods, such as METIS or other graph partitioners, which we exploit in the following.

Throughout the rest of the paper, we label the base cases with round-robin data distribution before and after the decomposition discussed in Section III-C as **RR** and **RR-splitLoc**, respectively. We label the cases using data distribution based on graph partitioning before and after the decomposition as **GP** and **GP-splitLoc**, respectively.

The distribution of work in the location computation phase, shown in Figure 3(d), is highly skewed as some locations have far more visitors than others. The objective of graph partitioning here is to minimize the communication between the computation phases subject to load balancing constraints for both phases. However, when we apply graph partitioning to our data, we observe significant variance in the load distributed between partitions and violations of balancing constraints. We investigate analytically and empirically how this affects scaling in the rest of this section. We first show how load balance is bounded by the granularity of work load. Then, we analyze how this affects scaling.

As discussed before, our input data is represented as a bipartite graph with persons and locations as nodes. Formally, we define this graph as $G=<V_\phi, V_\lambda, E>$, where V_ϕ is the set of person type vertices, V_λ is the set of location type vertices, and E is the set of edges. We begin our analysis with the following assumptions for the simplicity of discussion focusing on the location type vertices:

1) The number of location vertices ($v \in V_\lambda$) with degree d follows a power-law distribution given by $f = D \cdot prob(d) = D \cdot c \cdot d^{-\beta}$, where D is the location data size $|V_\lambda|$, c is a scaling constant such that $c \cdot \sum_1^\infty d^{-\beta} = 1$, and $\beta > 1$ is the power-law exponent.
2) The computational complexity of the work of a vertex $v \in V_\lambda$ is $O(d_v)$, where d_v is the vertex degree.
3) The computational load l_v of a vertex $v \in V_\lambda$ is approximated by $l_v = \alpha \cdot d_v + \gamma \approx \alpha \cdot d_v$, where α is a model constant.

We denote the maximum of vertex loads $\max(\{l_v | v \in V_\lambda\})$ by l^{max}, the maximum of vertex degrees $\max(\{d_v | v \in V_\lambda\})$ by d^{max}, and the average of vertex degrees by d^{avg}.

Suppose we have a K-way partition of V_λ, $P = \{p_i | i = 1,\ldots,K\}$, where $V_\lambda = p_1 \cup \ldots \cup p_K$. We define the load of a partition $p \in P$ as $L_p = \sum_{v \in p} l_v \approx \alpha \cdot \sum_{v \in p} d_v$. L^{tot} denotes the total sum of loads in V_λ. We define the maximum load of partitions L^{max} as $\max(L_p)$ where $p \in P$, the most heavily loaded partition p^{max} as $\text{argmax}_p(L_p)$, and the estimated upper bound for the speedup S_{ub} as L^{tot}/L^{max}. We consider this as the upper bound since the effect of communication and the scaling of the person phase are not taken into account. S_{ub} is further bounded by L^{tot}/l^{max} as $l^{max} \leq L^{max}$. Figure 4 shows S_{ub} as a result of graph partitioning. The general trend is that the larger the data, the higher the S_{ub} is. Although Figure 3(d) shows a similar load distribution in log-scale between IA and AR, l^{max} of AR is more than twice of that of IA, as shown in Table II.

Table II
THE TOTAL LOAD L^{tot} AND THE MAXIMUM LOAD PER LOCATION BEFORE (l^{max}) AND AFTER (ℓ^{max}) GRAPH MODIFICATION.

$\times 10^3$	CA	NY	MI	NC	IA	AR	WY
L^{tot}	545577	282940	151206	135180	43949	42319	7818
l^{max}	254.9	347.5	234.1	160.4	45.3	97.3	32.6
ℓ^{max}	2.9	2.0	2.2	2.0	1.9	1.7	1.6

As $L^{tot}/l^{max} = (\alpha \cdot \sum_{v \in V_\lambda} d_v)/(\alpha \cdot d^{max})$, $\log(S_{ub}) \lesssim \log(d^{avg} \cdot D) -$

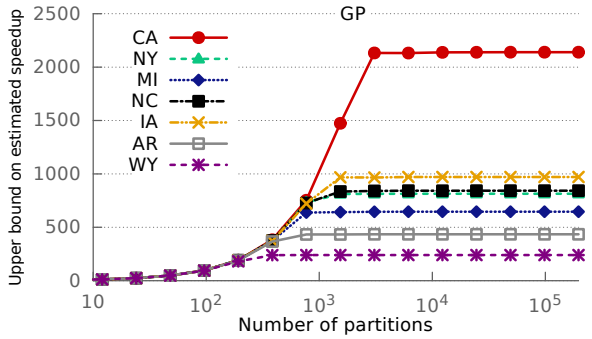
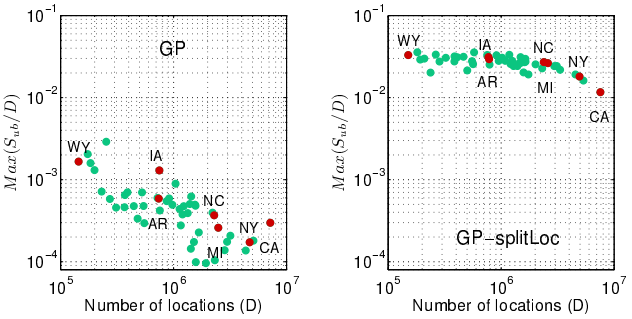


Figure 4. The estimated speedup for location computation based on the load distribution across data partitions. It is evaluated for each of seven states over various number of partitions between 12 and 196,608. (GP)

$\log(d^{max})$. From the power-law relationship, it is likely only one or very few vertices with d^{max} exist. By approximating this value f as 1 for d^{max}, we obtain $\log(cD\cdot(d^{max})^{-\beta})=\log(1)$, and thus $\log(d^{max})=\log(cD)/\beta$. This results in:

$$\log(S_{ub}/D) \lesssim \log(d^{avg}) - \frac{1}{\beta}\cdot\log(D) - \frac{1}{\beta}\cdot\log(c)$$

Note that $\log(d^{avg})\ll\log(D)$. Unless we compare extremely different sizes of D, d^{avg} is roughly the same, especially when $\beta\gg2$ and D is large. We observe that d^{avg}=14.35 and σ=1.69 in our US population data. Figure 5(a) shows that the scalability S_{ub}/D is reduced as the data size increases. In our population data, the peak load grows as the graph size increases. Section III-C describes our decomposition strategy to break the heavy-tailed structure in our graphs.

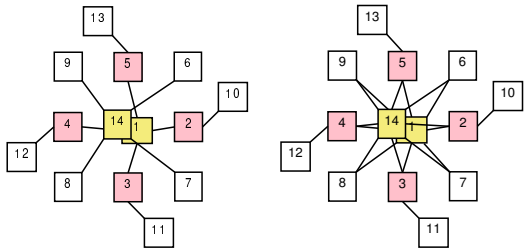


(a) Before decomposition (b) After decomposition

Figure 5. The maximum of the estimated speedup per location (S_{ub}/D) over the various number of partitions between 12 and 98304. Each dot represents one of 48 contiguous US states and DC.

C. Graph Preprocessing to Split Heavily-loaded Nodes

We develop a domain decomposition strategy to preprocess a graph before applying an existing partitioning strategy, such that the structure of the processed graph is closer to what the partitioner is designed for. During preprocessing, we modify the graph, taking advantage of unexploited extra parallelism in the application to split the work units (vertices) with the highest loads. For example, node 1 in Figure 2(a) can be split into two: node 1 and node 14 shown in Figure 6. Ideally, the split work units have no overlapping dependency between them as in Figure 6(a). This not only splits the work load, but also divides the communication, which helps reduce the maximum of both load and degree of vertices. Depending on the dependency pattern in the application, a split work unit can either retain the entire set of edges as in Figure 6(b) or require additional communication within the split pair.



(a) Divide edges (b) Retain edges

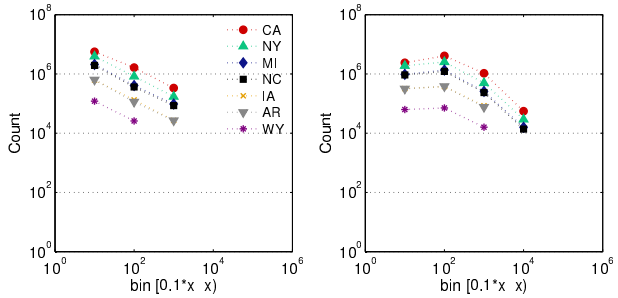
Figure 6. Two methods to split a heavily loaded node depending on available extra parallelism

In EPISIMDEMICS, the interaction between people defines the dependence. People only interact when they are present in the same sublocation. This allows us to split locations without adding extra communication edges as shown in Figure 6(a). We split a heavy location into multiple locations, each of which contains an exclusive subset of sublocations of the original location. Heavy locations are split, instead of being distributed by sublocation to keep the size of the graph minimal for scalability and partitioning efficiency, and to efficiently maintain shared data. In the future, we plan to model inter-sublocation mixing within a location, such as elevators and hallways, as in Figure 6(b). Even in such a case, we can split the load by dividing the susceptibles while replicating the infectious in a redundant sublocation.

To determine how heavy a location is, we rely on a platform-independent approximation instead of the platform-dependent load model defined for and used in graph partitioning in Section III-A and III-B. This approximation is used to split locations in order to bound the location load. We first define a weight for each sublocation type. Then, we add up the sublocation weights in the location. The sublocation weight is defined as the average number of visits to the sublocation. As we are interested in the heaviest locations, we determine the sublocation weight based on the largest locations from each state in terms of the number of sublocations. Then, we divide locations heavier than a threshold as evenly as possible. We determine the threshold based on the total load in the graph, the maximum number of partitions to use, and the largest weight of a sublocation.

As a result, the distribution of degree and load in the original graph shown in Figure 3(c) and (d) become the ones shown in Figure 7(a) and (b), respectively. The upper-bounds on the estimated speedup shown in Figure 4 improve to the ones in Figure 8. L^{tot}/l^{max} increases by a factor of, on average 89 (maximum 290, minimum 11) over 48 contiguous states and DC. Figure 5(b) shows the resultant improvement on S_{ub}/D. The modification reduces d^{max} by a factor of, on

average, 54 (maximum 341, minimum 12), while increasing D by at most 5.25%.



(a) Degree distribution (b) Static load distribution

Figure 7. Degree distribution and load distribution after graph modification. (GP-splitLoc)

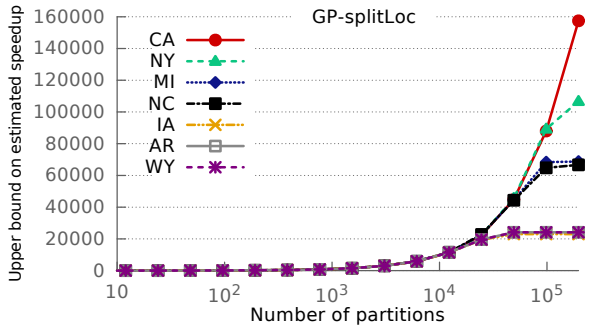


Figure 8. The estimated speedup evaluated identically as in Figure 4 after application-specific decomposition. (GP-splitLoc)

IV. Communication Optimizations

In addition to our algorithmic optimizations and data locality improvements through our new graph preprocessing scheme, we also extensively optimize EpiSimdemics compared to its first implementation in Charm++, including memory footprint and conditional branch reduction. Most importantly, we optimize the communication performance by exploiting hierarchical multi-core nodes, using advanced completion detection to speed up synchronization, and reducing buffering overhead and message size, all described in the following subsections. Combined, these optimizations provide an additional 40% reduction in execution time, shown as the difference between **RR no-opt** and **RR** in Figure 12. We rely on the optimized version to study the impact of social network input data on scalability.

A. Charm++ SMP Mode

We leverage Charm++'s SMP machine layer [9], which instead of creating one OS process per core of an n core node, the runtime creates k OS processes per node, such that $k<n$ and n/k is an integer. This allows chares within a process to leverage more efficient intra-node communication via shared memory for the following benefits: (i) inter-thread communication can be implemented with direct memory copy, (ii) the communication thread minimizes the interference between application compute functions and communication, (iii) sharing of read-only data across all threads reduces memory consumption. To enable this mode, Charm++ spawns a separate communication thread in each of the k OS processes in addition to the $n/k-1$ compute threads and then maps each thread to a separate core. The disadvantage of this approach is that it reduces the number of compute threads per node, since k cores are used as communication threads. However, the communication intensive nature of EpiSimdemics significantly benefits from the availability of a dedicated communication thread to offload messaging, leading to an overall increase in performance.

B. Completion Detection Synchronization

As discussed in Section II, there is a need for global synchronization at the end of each phase of the simulation. After each person has sent its visit messages to locations, a global barrier must be enforced before locations can start computing infections. However, since the individual location chares have no prior knowledge of how many messages to receive and from whom, a simple barrier is not adequate. Instead, we need a mechanism to detect the condition when there are no messages awaiting processing or in transit.

For this purpose, Charm++ provides a *Quiescence Detection (QD)* feature, which detects the global quiescence condition [10]. However, this approach requires global quiescence across the entire application. Since, in the future, we will use EpiSimdemics to perform multiple simulations simultaneously, using dynamic replication of state (chare arrays), we require an approach that enables us to perform synchronization local to a module.

We therefore rely on a novel *Completion Detection (CD)* [10] mechanism which can be applied to subsets of chares as long as the number of candidate producers is known apriori, which is given in our case. Completion is detected when the participating objects have produced and consumed an equal number of messages globally.

C. Message Aggregation

Prior versions of EpiSimdemics have shown that message aggregation is crucial to achieve good performance, even with a primitive flushing approach. However, this becomes ineffective when buffering space is exhausted or a receiver cannot keep up with an enormous volume of inbound messages. In this new version of EpiSimdemics, we expand on this and provide a novel built-in message aggregation mechanism to cope with such challenges efficiently[1]. It is used in PMs when sending visit messages to LMs, since this can lead to

[1]Note that the Charm++ team is currently working on TRAM (Topological Routing and Aggregation Module), which implements an application agnostic message aggregation in the runtime—however, this module was not available prior to the generation of most of the results presented here, and we are not yet able to determine to what degree it can replace our application-aware strategy.

a large number of small (36 byte) messages between two chares, caused by people visiting locations. Without message aggregation, we send each message upon generating it, by passing it as a parameter to the remote entry method on the destination chare. With our message aggregation, we do not send each individual message immediately after generation. Instead, we temporarily store messages in buffers organized per destination. Each source chare manages a set of buffers containing outgoing messages for a unique destination chare.

Our aggregation approach offers two flushing mechanisms controlled by runtime parameters. The per-buffer flushing ensures that a single buffer does not hold more than a user-defined number of outgoing messages. The space-wise flushing limits the memory footprint such that the buffer space of a chare does not grow beyond a user-defined size.

To avoid saturating network resources, message aggregation can be set to switch between sending and receiving regularly. Without this, the outstanding inbound messages remain in the message queue managed by the runtime system and may exhaust the queue space allocated by the runtime. When interleaving is not sufficient to avoid the resource saturation, we can further group multiple person managers and serialize message generation within a group. However, if possible, it is best for performance to generate all the messages without switching between sending and receiving.

V. Performance Results

All of our experiments were performed on the NSF Blue Waters Machine at NCSA. Blue Waters is an appropriate platform for EpiSimdemics due to its unusually large memory per node, high memory and communication bandwidth, and large number of cores. Blue Waters is a Cray XE6/XK7 installation at NCSA. Our experiments were confined to the 22,640 XE nodes. Each XE node has two AMD 6276 Interlagos processors, one Gemini NIC connected in a 3D torus and 64 GB of memory with 102.4 GB/s of memory bandwidth. Each Interlagos processor consists of eight Bulldozer core modules, with two integer units and one floating point unit per module, resulting in 16 core modules per node or 362,240 for the entire XE machine. It is possible to issue two threads per core module by issuing one thread per integer unit. However, we experimentally determined that the best performance for EpiSimdemics is achieved with one thread per core module.

We use Charm++ (version 6.4) built for gemini_gni-craxyxe, hugepages (8MB), and smp. For all the experiments, we fix the processor affinity of Charm++ processes and threads to prevent OS from moving them. We use GNU programming environment 4.1.40. The Cray balanced injection feature was not enabled.

A. Scaling Analysis

1) Charm++ Runtime and Optimizations: While EpiSimdemics is data-intensive and requires high memory and communication bandwidth, the visit message communication is not latency-sensitive. EpiSimdemics benefits from a system configuration that handles intensive network I/O and avoids memory access contention, while utilizing all core modules to achieve the best scaling. There are a total of four NUMA regions per XE6 compute node and four core modules per region. We identified the optimal configuration, which has one process per NUMA domain and one thread per core module (three worker threads per process), based on experimental results shown in Figure 9.

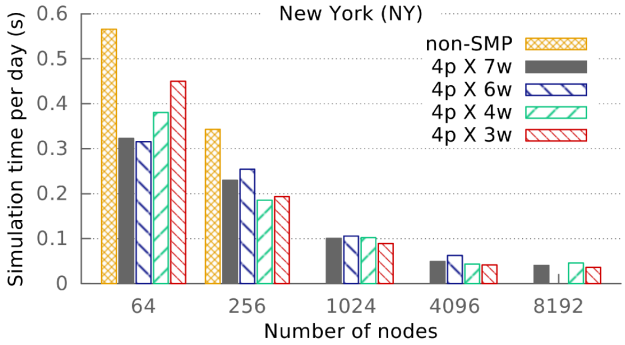


Figure 9. Identifying the best Charm++ SMP configuration on Blue Waters (RR-splitLoc). *n*p*X*m*w denotes the case where there are *n* smp processes per compute node and *m* worker threads per process.

2) Application-specific Data Partitioning: Figure 10 shows a breakdown of execution time for each core. The timeline plots show two iterations of EpiSimdemics from day 25 and day 26, while running MI data distributed and decomposed with four different methods, discussed in Section III, over 4K core modules (3072 cores). Due to space limitations, we plot the results for a uniformly sampled subset of 384 cores. Both RR and GP show extreme variance in location load distribution. This is because the distribution of load per work unit without decomposition is skewed as shown in Figure 3(d). GP improves data locality, leading to faster message generation than RR. RR-splitLoc as well as GP-splitLoc shows significant improvement over both RR and GP by reducing the load granularity.

Figure 11 shows the number of bytes received per core over six iterations from day 25 to day 30 from the same tracing runs shown in Figure 10. With GP, there is a peak exceeding 12 MB, which corresponds to the peak orange bar in Figure 10. This is consistent with the load model discussed in Section III-A, in which the load is roughly proportional to the vertex degree. This occurs because the particular processor has the location with the maximum load without decomposition. The average volume of communication per core is less than that of RR, which is around 4 MB. This is due to the fact that the graph partitioner does not balance communication evenly, but only minimizes its total volume.

Figure 12 shows the results of the largest scale experiment that was run. We used 352K core-modules (using 99.5% of the XE6 compute nodes) and achieved a speedup of 58,649 for an efficiency of 16.3% using the entire US population data.

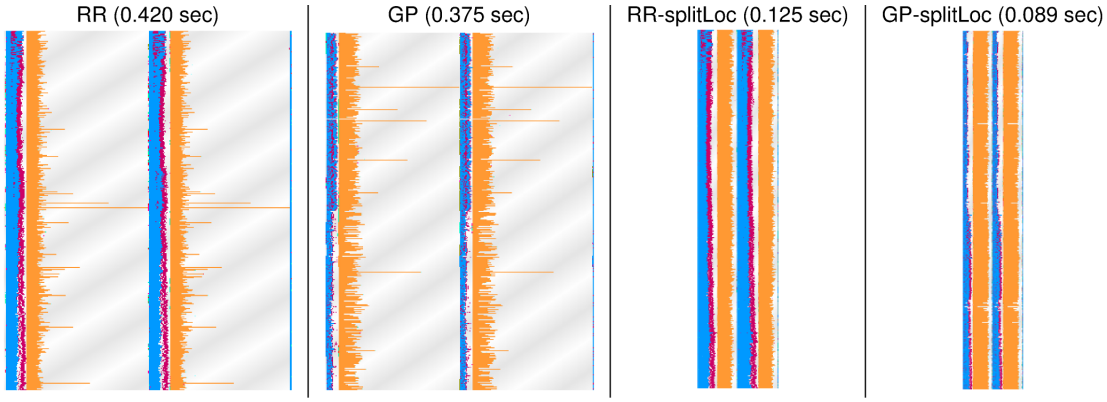


Figure 10. Timeline plots of two iterations of EPISIMDEMICS using MI data and 4K core modules on Blue Waters (sampled to show 384 compute PEs). The blue component shows the person computation, the red shows the receiver side message handling, and the orange shows the location computation. The time above the plot shows the average time taken over the two iterations while running the CHARM++ Projections visual analysis tool.

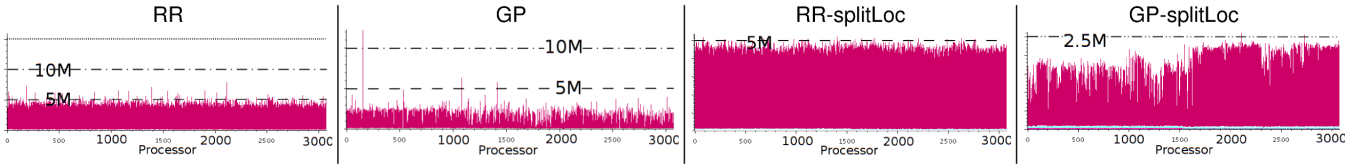


Figure 11. Amount of data received per PE in MB over six iterations using MI data and 4K core modules. GP has the highest maximum communication volume while GP-SPLITLOC has the lowest. Both have smaller average communication volume than RR and RR-SPLITLOC.

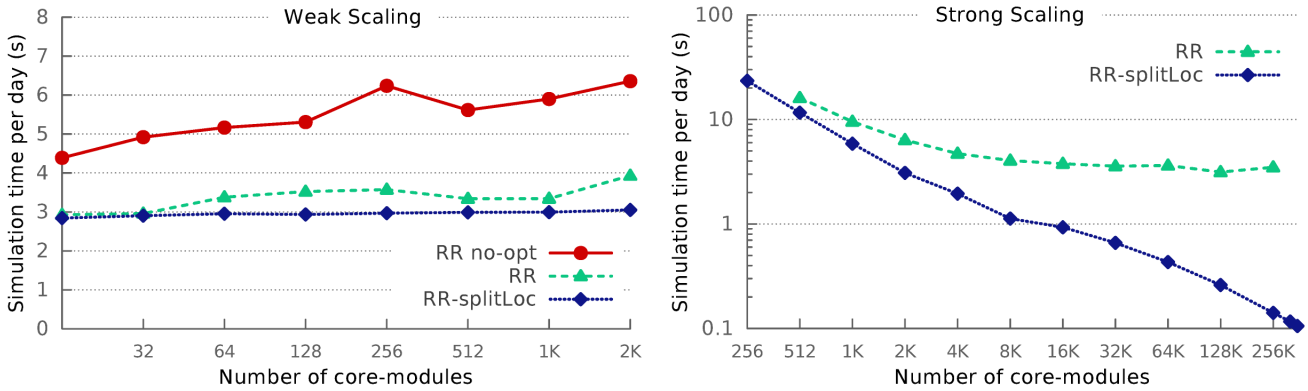


Figure 12. Weak and strong scaling performance of EPISIMDEMICS on Blue Waters. For weak scaling, we choose a subset of the continuous US by state such that the number of people per processor remains approximately constant. With the largest dataset fixed, we demonstrate weak scalability by using less than 1/4 of the available memory, which allows us to run up to 2k core modules. For strong scaling, we use data for the whole continental US.

These numbers represent the best scaling for an individual-based epidemic diffusion simulation by a factor of 5.3.

Figure 13 shows the strong scaling of EPISIMDEMICS with state data listed in Table I. This shows the effectiveness of the SPLITLOC decomposition discussed in Section III-C. The general trends compared to the estimated cases shown in Figure 4 and Figure 8 are as follows. Without decomposition, both the actual speedup and the estimated one reaches the peak and stops increasing at some point while increasing the number of processors. With decomposition, we observe that the speedup with CA, NY, MI, and NC keeps increasing up to 128K core modules used, and that with IA, AR, and WY keeps increasing up to 64K core modules. The peak speedups achieved by the actual runs without using decomposed data are on average 75% of that of the estimated speedup (S_{ub}), maximum 92%, and minimum 50% among the seven states' data. On the other hand, the peak speedups achieved using decomposed data are quite low, on average 6.3%, maximum 9%, and minimum 4.5%. The estimated speedup with decomposed data up to 128K core modules is close to the ideal. This suggests that we removed one important bottleneck, but uncovered others.

Another identified bottleneck is the unbalanced communication shown in Figure 11. Figure 14 shows the inbalance in the distribution of edges. Each data point represents the maximum per-partition edge cut as a result of graph partitioning. The line of matching order represents the ideally balanced all-remote-communication case, which is calculated

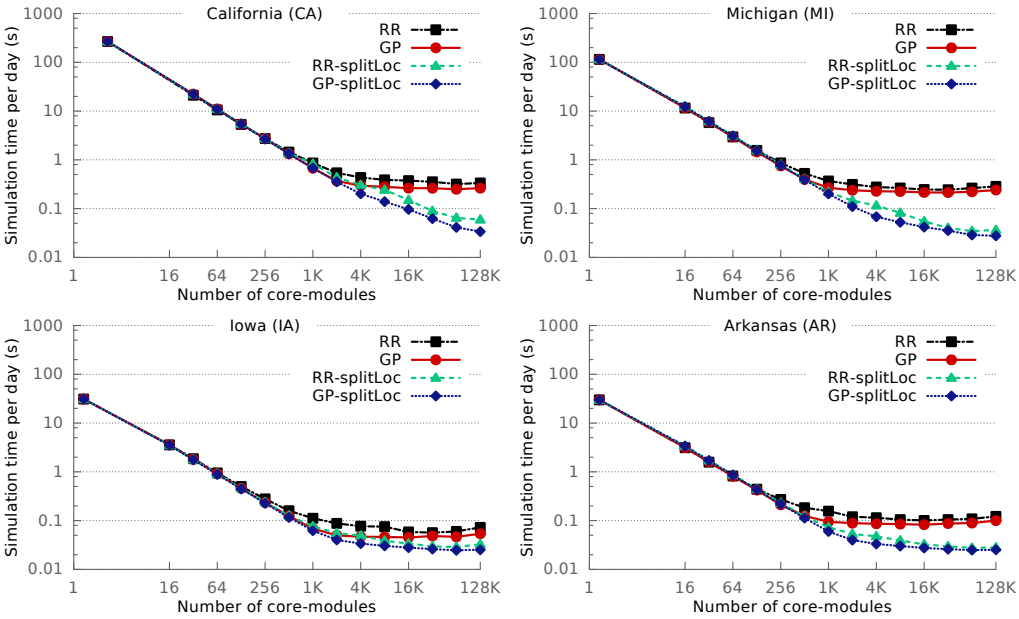


Figure 13. Strong scaling performance of EPISIMDEMICS for selected states on Blue Waters

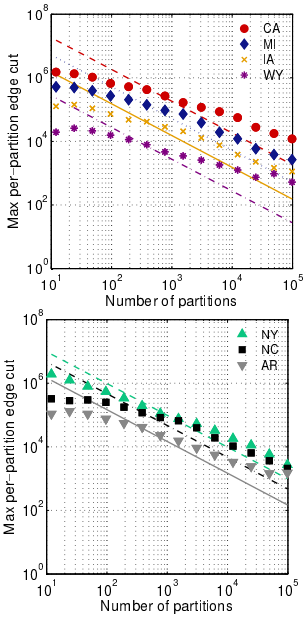


Figure 14. The maximum per-partition edge cut (GP-splitLoc).

as the number of total edges divided by the number of partitions hypothetically imagining that all edges are cut. With WY, the maximum per-partition edge cut is 19 times larger than the all-remote-communication case with 98,304 data partitions. On the other hand, with NY data, the ratio is 2.7. The average ratio across all seven states is 7.83.

VI. RELATED WORK

Among many agent-based epidemiological platforms are those developed by Eubank et al. [11], Longini et al. [12], Ferguson et al. [13], and Parker et al. [14]. The system described in [13] is implemented for shared memory platforms and, thus, is limited by the amount of available shared memory. The works in [12] and [14] either use structured social contact networks that are more amenable to efficient parallel computation, but which, arguably, are less representative of real-world social networks, or lack the rich set of interventions required to accurately model real-world responses to pandemics. In any case, none of these have been shown to scale to more than a thousand or so cores.

The work by Permualla et al. [4] has the best scaling of any individual level Epidemiology simulation known to the authors, and is based on previous EPISIMDEMICS work [1]. The system uses the same disease model and transmission function, but is based on a hierarchical social network construction similar to [12], a population of homogeneous agents, and lacks the ability to model interventions.

Traditional graph partitioning tools primarily focus on minimizing total edge cuts while enforcing load balance as a constraint rather than an optimization target [8, 15, 16]. Such partitioning methods have been most successful in areas as mesh-based PDE simulation, VLSI layout design, and sparse matrix decomposition [16, 17]. Although minimizing the total edge cuts limits the maximum edge cuts per partition, these tools do not balance edge cuts across partitions, which is also important for minimizing communication cost.

Partitioning extremely large, highly irregular data is left as an open problem. Abou-Rjeili et al. [18] propose a new clustering-based coarsening scheme for power-law graphs, which identifies and collapses groups of vertices that are highly connected. The method described in [19] divides the vertices of a graph into k almost equal groups such that the sum of the weight of the edges connecting vertices in different partitions is minimized. Pearce et al. [20] evenly divide sorted edge lists to reduce communication hotspots while accommodating high degree vertices over multiple partitions using ghosts. Our approach is different from these as we enhance application-specific load balancing with locality optimization. We split nodes with heavy computation and then utilize graph partitioning for locality optimization.

VII. FUTURE WORK

The work load in EPISIMDEMICS contains both deterministic and non-deterministic portions. We focus on the former in this paper, and are currently investigating the latter. CHARM++ runtime offers measurement-based load balancing (LB) framework based on the *principle of persistence*. Since our application can have highly dynamic computation, this is not sufficient. Our plan is to address the dynamism by the application-specific prediction of work load. The goal is to avoid incurring excessive overhead by initiating LB phases without a sufficient gain in performance as in [21], but by using application-specific information.

VIII. Conclusions

Contagion simulations play an important role in understanding and combating epidemics. The need to model national and multinational regions with increasing accuracy, while adhering to strict deadlines, requires large computing resources. The extreme irregularity in the underlying graph data makes scaling contagion simulations challenging. In this work, we presented advances in a large contagion simulation framework, EpiSimdemics, that allowed us to significantly improve the performance and scalability of this application.

Many advancements were made in order to achieve this result, taking advantage of application semantics at both the algorithmic and implementation level. These include domain decomposition and effective partitioning of the person-location graph to minimize the impact of the heavy-tailed load distribution of the graph data, and communication optimizations through message aggregation, efficient symmetric multiprocessing (SMP) and optimized synchronization.

As a result of these optimizations, we have shown unprecedented scaling of an individual-based contagion diffusion model, scaling to over 352K cores on Cray XE6, a five-fold increase over the previous state of the art on Cray XT5 while still increasing parallel efficiency compared to the largest prior run at 64K cores. The improved turn around times will have a positive impact on the ability of policy makers to respond to emerging pandemics in the future.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-648533). This work has been partially supported by NSF Grants PetaApps OCI-0904844 and NetSE CNS-1011769, DTRA Grant HDTRA1-11-1-0016, DTRA CNIMS Contract HDTRA1-11-D-0016-0001, and NIH MIDAS Grant 2U01GM070694-09.

This research is part of the Blue Waters sustained-petascale computing project, which is supported by NSF award number OCI 07-25070 and the state of Illinois. Blue Waters is a joint effort of the University of Illinois and NCSA. This work is also part of the Contagion PRAC allocation support by NSF award number OCI-0832603.

References

[1] 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 *Proc. of 2008 ACM/IEEE conference on Supercomputing*, 2008.

[2] NDSSL, "Case studies," http://ndssl.vbi.vt.edu/sc13/CaseStudies.html.

[3] L. Kalé and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proc. of OOPSLA'93*, Sept. 1993, pp. 91–108.

[4] K. S. Perumalla and S. K. Seal, "Discrete event modeling and massively parallel execution of epidemic outbreak phenomena," *Simulation*, vol. 88, no. 7, pp. 768–783, Jul. 2012.

[5] Christopher L. Barrett et al., "Generation and analysis of large synthetic social contact networks," in *Proc. of 2009 Winter Simulation Conference*, Dec. 2009.

[6] K. Bisset, X. Feng, M. Marathe, and S. Yardi, "Modeling interaction between individuals, social networks and public policy to support public health epidemiology," in *Proc. of 2009 Winter Simulation Conference*, Dec. 2009.

[7] Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong, "Analysis of topological characteristics of huge online social networking services," in *Proc. of 16th international conference on World Wide Web*, ser. WWW '07, 2007, pp. 835–844.

[8] G. Karypis and V. Kumar, "Multilevel Algorithms for Multi-Constraint Graph Partitioning," in *ACM/IEEE Conference on Supercomputing*, 1998, pp. 1–13.

[9] C. Mei, Y. Sun, G. Zheng, E. J. Bohm, L. V. Kalé, J. C.Phillips, and C. Harrison, "Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime," in *Proc. of 2011 ACM/IEEE conference on Supercomputing*, Nov. 2011.

[10] *The Charm++ Programming Language Manual, (Version 6.4.0)*, Parallel Programming Laboratory, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 2012.

[11] S. Eubank, H. Guclu, A. Vullikanti, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang, "Modelling disease outbreaks in realistic urban social networks," *Nature*, vol. 429, no. 6988, pp. 180–184, May 2004.

[12] I. M. Longini, A. Nizam, S. Xu, K. Ungchusak, W. Hanshaoworakul, D. A. Cummings, and E. M. Halloran, "Containing pandemic influenza at the source," *Science*, vol. 309, no. 5737, pp. 1083–1087, August 2005.

[13] N. M. Ferguson, M. J. Keeling, W. J. Edmunds, R. Gant, B. T. Grenfell, R. M. Amderson, and S. Leach, "Planning for smallpox outbreaks," *Nature*, vol. 425, no. 6959, pp. 681–685, 2003.

[14] J. Parker and J. M. Epstein, "A distributed platform for global-scale agent-based models of disease transmission," *ACM Transactions on Modeling and Computer Simulation*, vol. 22, no. 1, Dec. 2011.

[15] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *Proc. of Intl. Conference and Exhibition on High-Performance Computing and Networking*. Springer-Verlag, 1996, pp. 493–498.

[16] Ümit V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. on Parallel and Distributed Computing*, vol. 10, pp. 673–693.

[17] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. 7, no. 1, pp. 69 –79, Mar. 1999.

[18] A. Abou-Rjeili and G. Karypis, "Multilevel algorithms for partitioning power-law graphs," in *IEEE Intl. Parallel and Distributed Processing Symposium*, 2006.

[19] S. Lin and X. Cheng, "Bc-ga: A graph partitioning algorithm for parallel simulation of internet applications," *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, vol. 0, pp. 358–365, 2008.

[20] R. A. Pearce, M. Gokhale, and N. M. Amato, "Scaling techniques for massive scale-free graphs in distributed (external) memory," in *IEEE Intl. Parallel and Distributed Processing Symposium*, 2013.

[21] H. Menon, N. Jain, G. Zheng, and L. V. Kalé, "Automated Load Balancing Invocation based on Application Characteristics," in *IEEE Cluster 12*, September 2012.