PREEMPT: Scalable Epidemic Interventions Using Submodular Optimization on Multi-GPU Systems

Marco Minutoli*, Prathyush Sambaturu†, Mahantesh Halappanavar*‡,
Antonino Tumeo* Ananth Kalyananaraman‡* and Anil Vullikanti†
*Pacific Northwest National Laboratory, Richland, WA; Email: {marco.minutoli, hala, antonino.tumeo}@pnnl.gov
†University of Virginia, Charlottesville, VA; Email: pks6mk@virginia.edu, vsakumar@virginia.edu
‡Washington State University, Pullman, WA; Email: ananth@wsu.edu

***Abstract*—Preventing and slowing the spread of epidemics is achieved through techniques such as vaccination and social distancing. Given practical limitations on the number of vaccines and cost of administration, optimization becomes a necessity. Previous approaches using mathematical programming methods have shown to be effective but are limited by computational costs. In this work, we present PREEMPT, a new approach for intervention via maximizing the influence of vaccinated nodes on the network. We prove submodular properties associated with the objective function of our method so that it aids in construction of an efficient greedy approximation strategy. Consequently, we present a new parallel algorithm based on greedy hill climbing for PREEMPT, and present an efficient parallel implementation for distributed CPU-GPU heterogeneous platforms. Our results demonstrate that PREEMPT is able to achieve a significant reduction (up to $6.75\times$) in the percentage of people infected and up to 98% reduction in the peak of the infection on a city-scale network. We also show strong scaling results of PREEMPT on up to 128 nodes of the Summit supercomputer. Our parallel implementation is able to significantly reduce time to solution, from hours to minutes on large networks. This work represents a first-of-its-kind effort in parallelizing greedy hill climbing and applying it toward devising effective interventions for epidemics.**

***Index Terms*—Epidemic networks, vaccination, submodular, influence maximization, CPU-GPU, parallel algorithms.**

I. INTRODUCTION

Social distancing and vaccination are the primary strategies (or public health interventions) for controlling the spread of an epidemic [1]–[3]. Such interventions are very expensive to implement, as the current COVID-19 outbreak has already demonstrated. Therefore, designing optimal intervention strategies is a fundamental public health challenge. Due to the complexity of disease spread, large agent-based models have been used [2], [3]. These models are compartmental models that simulate a Susceptible-Infectious-Recovered (or SIR) or other similar stochastic diffusion processes, on a network (§II). The problem of designing an efficient intervention (which we refer to as the EPICONTROL problem) is equivalent to removing nodes (in the case of vaccination) or edges (in the case of social distancing) within a given budget, so that most lives are saved. For the purpose of this paper, we focus on the harder problem of vaccination; the methods extend to social distancing as well.

The EPICONTROL problem for vaccination-based intervention is NP-hard [4], and prior work either lacks any sound theoretical support (e.g., strategies which choose nodes based on degree or centrality [5]–[7]), or uses spectral graph theory methods [8]–[11], or math programming techniques [12]–[14], which give approximation guarantees, but can take hours to even days to complete on even moderate size graphs ($\sim 10^5$ nodes). Our focus is to find effective interventions that can scale to networks with well over millions of nodes.

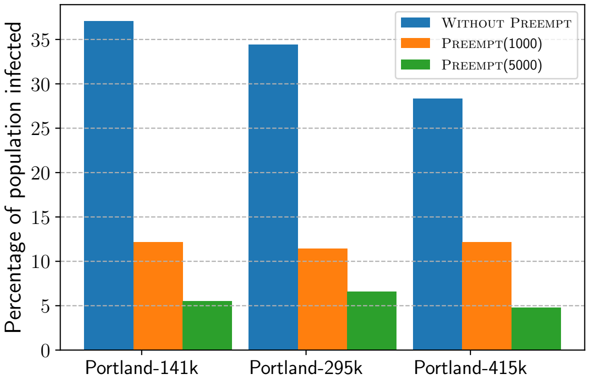


Fig. 1: A comparison of the percentages (%) of population infected with and without our proposed method PREEMPT, for three contact networks of Portland taken from [15], [16] (§V). Even with relatively low budgets for vaccination (1000 and 5000 nodes), we obtain anywhere between 2.61× to 6.75× reduction in the percentages of infected nodes over WITHOUT PREEMPT.

Contributions: In this paper, we present a novel formulation to address the EPICONTROL problem. Our formulation maps the problem to one of identifying an optimal set of nodes (or "seeds") for vaccination so that the effective number of infections on the network can be minimized. This formulation allows us to leverage principles from influence maximization—a well known problem in network science [17]. Based on this new formulation and provable properties, we devise an algorithm and present its parallel implementation.

More specifically, we make the following contributions:

- Starting from an observation that EPICONTROL is submodular on a forest, and the intuition that good vaccination strategies must have high reachability, we design our PREEMPT strategy based on influence maximization (§III). We show

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

that PREEMPT involves maximizing a submodular function.

- We propose a greedy hill-climbing based algorithm called PREEMPT-HC, to approximately implement PREEMPT, and yield a solution to EPICONTROL, for a given budget (§III-C);
- We present an efficient parallel algorithm targeted to run on distributed CPU-GPU heterogeneous platforms.
- Our experiments demonstrate strong scaling on networks with millions of vertices, on the current fastest supercomputer, Summit with CPU-GPU nodes; effectively reducing time to solution from hours to minutes on up to 128 nodes of Summit (§VI). The performance improvements are more pronounced for contact networks, owing to significant speedups achieved during the dominant step of the algorithm (e.g., up to 155×); and
- Our application on a synthetic social contact network for the population for Portland, OR, which has been used in public health analysis [15], [16] shows significant reductions in the number of infections achieved by our method PREEMPT, as shown in Fig. 1.

To the best of our knowledge, this work represents the first effort in parallelizing a simulation of optimal intervention methods for controlling the spread of epidemics using submodular optimization, which can scale to networks with millions of nodes. Our approach using hill climbing is motivated by a need to support scalable implementations that are better equipped to generate high quality seed sets, and to better handle dynamically evolving needs in seed selection [18] and incorporation of additional constraints (such as fairness) [19]. The work is also the first of its kind to scale greedy hill climbing algorithm on heterogeneous CPU-GPU platforms. Also to the best of our knowledge, PREEMPT type of strategies have not been studied for containing epidemics, especially in the context of uncertainties or lack of prior information—making the work relevant to researchers, practitioners, and decision and policy makers in the epidemics domain.

II. PRELIMINARIES

SIR process. We consider the SIR (Susceptible (S), Infected (I), Recovered (R)) model of epidemic spread on a network [17]. Let $G(V, E, \omega)$ be a directed graph representing the contact network. V is a set of vertices representing people; E is a set of edges representing direct contact between pairs of vertices modeling the spread of a disease, so that $(u, v) \in E$ represents an opportunity for an infection to spread from node u to node v; and, $\omega(.)$ is a weight function, with ω_{uv} denoting the conditional probability that node v gets infected, if u is infected[1]. In this model, each node $u \in V$ is in one of S, I or R states. The disease process starts when a small subset of nodes (sources) is in state I, and the rest of the nodes are in state S. A node u in state I attempts to infect each neighbor v in state S independently with probability w_{uv}; if the infection is successful, v switches to state I. Note that if node v has multiple infected neighbors u_1, u_2, then, v gets infected with probability $1 - (1 - w_{u_1v})(1 - w_{u_2v})$. For simplicity, here we assume an infectious duration of one time step, so a node in state I at time t switches to state R at time $t + 1$. The SIR process ends when all the nodes are in states S or R. Fig. 2 illustrates the SIR process on a contact network with five nodes, with a single node as the source.

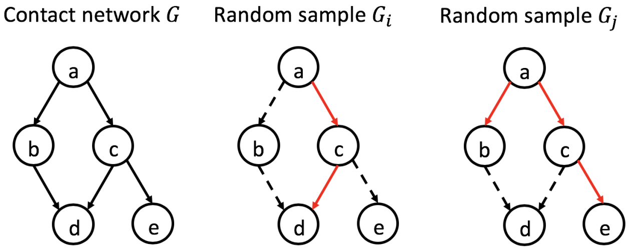


Fig. 2: *The SIR process* on a graph G, along with two samples (G_i and G_j) drawn from it. Consider G_i, in which a is initially in state I at time $t = 0$. At $t = 1$, c changes to state I, while a changes to R. At $t = 2$, d changes to state I, while c changes to state R. At $t = 3$, d also changes to state R. Red edges represent the corresponding random sample; for instance, G_i occurs with probability $w_{ac}w_{cd}(1 - w_{ab})(1 - w_{bd})(1 - w_{ce})$.

Diseases such as COVID-19, HIV, chicken pox, and dengue hemorrhagic fever, require additional disease states such as ”Exposed (E)” and ”Asymptomatic (A)”. The resulting models can accordingly be named as SEIR [20], SEIRS [21], and SEAIR [22]—as surveyed in [23], [24].

Table I summarizes some of the key notation used in this paper.

TABLE I: Notation used in this paper.

Symbol	Description
$G(V, E, \omega)$	Directed graph with edge weights
B	Initial set of infected nodes
S, k	Set of nodes to vaccinate and a corresponding budget
G_i	One of the η graph samples generated from G
$\sigma_G(B)$	Influence of set B in G
$\widetilde{\sigma}_{G_i}(B, S)$	Influence of B in G when S is removed
$\widetilde{\lambda}_{G_i}(B, S)$	Number of nodes saved (from B's influence) when S is removed from G

Random samples and influence. Following the work of Kempe *et al.* [17], we use the term “influence” to represent the infections caused by a node. Given G and η random numbers $r^1_{uv} ... r^\eta_{uv}$ for each edge in E (for a total of $\eta \times |E|$ random numbers), we generate η random samples of G, represented by set $SG = \{G_1, G_2, \ldots, G_\eta\}$, such that an edge (u, v) is part of G_i iff its weight $\omega_{uv} \geq r^i_{uv}$. Furthermore, we define $\sigma_{G_i}(B)$ as the total number of infected (influenced) vertices in G_i for a given set of initially infected vertices B. Formally,

$$\sigma_{G_i}(B) = \left| \bigcup_{b \in B} \mathcal{R}(b, V) \right|, \tag{1}$$

[1] If the input network is undirected, we treat each edge as a bidirectional edge with the same weight.

where $\mathcal{R}(b, V)$ is the set of vertices in V that can be reached when source vertex $b \in B$ is activated (infected) in G_i. For the example shown in Fig. 2, if $B = \{a\}$, then the number of infected nodes in G_i is $\sigma_{G_i}(\{a\}) = 3$. Henceforth, we refer to set B as the *source set*, and use $\mathbb{E}_{G_i}[\sigma_{G_i}(B)]$ to denote the expected number of infections due to the source set B.

Vaccination as an intervention to save nodes. In this paper, we consider interventions in the form of vaccinations. More specifically, a node that is in the *vaccinated* state can *no* longer participate in disease transmission with a probability of q—therefore, this is equivalent to removal of that node and all its incident edges from the network with the same probability. For the purpose of this paper, we use the setting of $q = 1.0$, although our approach itself can be extended to the arbitrary value setting. Let S denote a set of vaccinated nodes. As before, let B denote the source set (of initially infected nodes). We define a new metric, $\widetilde{\sigma}_{G_i}(B, S)$ that denotes the number of infections given B and S, as follows:

$$\widetilde{\sigma}_{G_i}(B, S) = \left| \bigcup_{b \in B} \mathcal{R}(b, S, V) \right|, \tag{2}$$

where $\mathcal{R}(b, S, V)$ is the set of nodes in G_i that can be reached from the source nodes (B) after all nodes in S are removed.

Given the above definitions, we define the *number of nodes saved* to be:

$$\widetilde{\lambda}_{G_i}(B, S) = \widetilde{\sigma}_{G_i}(B, \emptyset) - \widetilde{\sigma}_{G_i}(B, S), \tag{3}$$

and the resulting outbreak size (i.e., when S is vaccinated) is $\mathbb{E}_{G_i}[\widetilde{\sigma}_{G_i}(B, S)]$. In the example of Fig. 2, for random sample G_j, suppose $B = \{a\}$, and $S = \{c\}$. Then, $\widetilde{\sigma}_{G_j}(B, S) = 2$ (as $\mathcal{R}(a, S, V) = \{a, b\}$), and $\widetilde{\lambda}_{G_j}(B, S) = 2$.

III. Problem Formulation and Algorithms

In this section, we formulate the problem and prove related properties that will help us to subsequently design efficient algorithms (§III-C). Our main motivation is to determine the vaccination set S, that maximizes the number of nodes saved, subject to a budget constraint that $|S| \leq k$. This is formally defined as problem EpiControl below.

Definition III.1 (EpiControl)**.** *Given a graph G, a set of initially infected nodes B, and a budget k, find a set of nodes $S \subseteq V$ to vaccinate, such that $|S| \leq k$, and $\mathbb{E}[\widetilde{\lambda}_G(S)]$ is maximized.*

In §III-A, we show that EpiControl in rooted trees is submodular. However, maximizing the number of lives saved in a general graph is *not* submodular and therefore a harder optimization problem. Consequently, we aim to maximize the impact of a vaccination campaign. More specifically, we consider a preemptive scenario where at the anticipated onset of an outbreak in a country (or region), agencies need to decide which k nodes to vaccinate in order to minimize the expected spread of infections post-onset (or equivalently maximize the number of saved lives post-onset). If an arbitrary node v is to be saved, then it means *every path* that connects any infected node to v should have at least one vaccinated node along the way. In other words, all paths from the infected nodes to v need to be broken. The goal then becomes one of computing S, a target set of nodes to vaccinate, which collectively has the maximum reachability into the network.

We observe that this goal is indeed what the classical influence maximization formulation [17] also intends to address. More specifically, if we treat the set of vaccinated nodes S as the initial set of "activated" nodes, then a solution for influence maximization can help us identify the maximum influential seed set of size $\leq k$ that can become targets for a preemptive vaccination campaign. In other words, this strategy would allow us to leverage existing solutions for influence maximization for devising a preemptive solution to EpiControl. We label this strategy as Preempt and formulate it the following manner:

Definition III.2 (Preempt)**.** *Given a graph G and a budget k, determine a set of nodes $S \subseteq V$ to vaccinate such that $\mathbb{E}[\sigma_G(S)]$ is maximized, and that $|S| \leq k$, treating S as the initial set of activated nodes.*

We provide the algorithmic details for Preempt in §III-C and empirical evaluation in §VI. In what follows, with the goal of designing efficient greedy solutions for the above defined optimization problems, we aim to prove the submodularity property for a specific instance of EpiControl, and for Preempt. Submodularity can be defined as follows:

Definition III.3 (SubModular)**.** *Let S be a finite set. A function $f\colon 2^S \rightarrow \mathbb{R}$ is submodular if for any subsets $X \subseteq Y \subseteq S$ and $x \in S \setminus Y$, $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$.*

Simply put, the function f is such that adding x into a smaller subset X is expected to generate a larger gain than adding x into a larger set Y that contains X. Therefore, this gives us a greedy incentive to find and add the best possible x at any given point during an incremental seed set construction.

In what follows, we prove a submodular function for EpiControl for the special case when G_i is a rooted tree, and subsequently prove the property for the generalized setting for Preempt.

A. Submodularity for EpiControl in Rooted Trees

Theorem III.1. *If G_i is a rooted tree, then $\widetilde{\lambda}_{G_i}(B, S)$ is a submodular function of S.*

Proof. Recall from Eqn. 3 that $\widetilde{\lambda}(B, S)$ is the number of nodes saved[2] (from the influence of B) when nodes in S are vaccinated (i.e., removed from G_i). We use $\Lambda(u)$ to denote the set of nodes in the subtree rooted at any node u (inclusive of u). Let us consider subsets X and Y such that $X \subseteq Y \subseteq V$, and a node $x \in V \setminus Y$.

If x is added to X, then $\widetilde{\lambda}(B, X \cup \{x\}) - \widetilde{\lambda}(B, X) \leq |\Lambda(x)|$. Specifically, let $u \in \Lambda(x) \cap X$—i.e., any node in the subtree

[2] For convenience, we ignore the subscript corresponding to the graph sample in this proof.

of x that already exists in X. We define $U_{x,X} = \bigcup_u \Lambda(u)$. Then, $\widetilde{\lambda}(B, X \cup \{x\}) - \widetilde{\lambda}(B, X) = |\Lambda(x)| - |U_{x,X}|$.

Alternatively, let us evaluate the impact of adding x to Y, by considering the corresponding term $U_{x,Y}$ for Y. Since any node $u \in X$ is also a member of Y, $|U_{x,Y}| \geq |U_{x,X}|$. It follows, $\widetilde{\lambda}(B, X \cup \{x\}) - \widetilde{\lambda}(B, X) \geq \widetilde{\lambda}(B, Y \cup \{x\}) - \widetilde{\lambda}(B, Y)$. □

Since $\widetilde{\lambda}_{G_i}(B, S)$ is submodular in this case and non-negative linear combinations of submodular functions are still submodular, it follows (as in [17]) that $\mathbb{E}[\widetilde{\lambda}_{G_i}(B, S)]$ is submodular, and a greedy algorithm gives a constant factor approximation to EpiControl. However, as can be noted, the proof for Theorem III.1 is predicated on the assumption that G_i is a rooted tree. If not, then there could exist additional paths to node u that bypass x thereby breaking the submodular guarantee. This motivates the idea of Preempt.

B. Submodularity for Preempt *in Graphs*

We now show that Preempt in a general graph is submodular. Let G be an arbitrary contact graph and G_i be one of its random samples.

Theorem III.2 (Submodularity theorem)**.** *Given a graph G, a budget k and diffusion process based on the SIR model, the optimization function of* Preempt*, which is to maximize the influence of set S in G, is submodular, and $S \leq k$.*

Proof. Recall from Eqn. 1 that $\sigma_{G_i}(S)$ is the number of all reachable nodes in G_i (i.e., $\mathcal{R}(s, V)$) from all nodes $s \in S$. Consider any two subsets X and Y such that $X \subseteq Y \subseteq V$, and a node $s \in V \setminus Y$. If s is added to X, then the number of *new* nodes added will be that subset of $\mathcal{R}(s, V)$ that are *not* already included in $\bigcup_{x \in X} \mathcal{R}(x, V)$. Now, consider the contribution of adding s to set Y. Since $X \subseteq Y$, the number of new nodes contributed by the addition of s to Y can be *at most* as large as the number of nodes in $\mathcal{R}(s, V)$ that are *not* already included in $\bigcup_{x \in X} \mathcal{R}(x, V)$. Thus, $\sigma_{G_i}(X \cup \{s\}) - \sigma_{G_i}(X) \geq \sigma_{G_i}(Y \cup \{s\}) - \sigma_{G_i}(Y)$.

The expected number of nodes activated in G is given by the weighted average over all the samples:

$$\mathbb{E}[\sigma_G(S)] = \sum_{G_i \in SG} \text{Prob}[G_i] \cdot \sigma_{G_i}(S). \tag{4}$$

Since a non-negative linear combination of submodular functions is also submodular, $\mathbb{E}[\sigma_G(S)]$ is submodular. □

As shown in Kempe *et al.*, the influence maximization problem is NP-hard for the independent cascade model [17]. We therefore present approximation algorithms for Preempt.

C. A Greedy Algorithmic Framework for Preempt

We present a greedy strategy to exploit the submodular property of Preempt. We begin with the greedy approximation algorithmic framework based on influence maximization, as summarized in Algorithm GreedyIM. The algorithm begins with an empty set for S and iterates until k seeds have been identified by selecting a vertex (s_{best}) at each step (Line 3) that provides the maximum marginal gain for the influence function, $\sigma(\cdot)$. Algorithm GreedyIM can be shown to be approximate if $\sigma(\cdot)$ is a non-negative monotone submodular function (§III).

Algorithm 1: GreedyIM: Selects a set of nodes S of size at most k that maximize the influence, $\sigma(\cdot)$.

```
Input  : $(G = (V, E, \omega), k)$
Output: $S$
1 $S \leftarrow \emptyset$
2 while $|S| < k$ do
3     $s_{\text{best}} \leftarrow \arg\max_{s \in V \setminus S}(\sigma_G(S \cup \{s\}) - \sigma_G(S))$
4     $S \leftarrow S \cup \{s_{\text{best}}\}$
```

Theorem III.3 (Approximation Bound [17], [25], [26])**.** *Let S be the set of size k computed iteratively by Algorithm* GreedyIM *by selecting the vertex with largest marginal gain in the value of a non-negative monotone submodular function $\sigma(\cdot)$. Let S^* be the set that maximizes the value of $\sigma(\cdot)$ over all possible set of k elements in G. Then, $\sigma(S) \geq (1 - 1/e) \cdot \sigma(S^*)$.*

We refer the reader to [17] for the proof of Theorem III.3. The approximation guarantee of $(1 - 1/e)$ relies on the fact that GreedyIM can compute the exact value of $\sigma(\cdot)$, which is hard to compute in a probabilistic setting.

An alternative approach is to use multiple simulations of the diffusion process and sampling to provide close approximations of $\sigma(S)$. To this end, the *Greedy Hill Climbing algorithm* of Kempe *et al.* [17] becomes relevant, as it provides an approximation guarantee of $(1 - 1/e - \epsilon)$, for any $\epsilon > 0$. This algorithm is known to be computationally more expensive compared to modern algorithms such as IMM from Tang *et al.* [27], that are applicable to specific models of diffusion such as SIR. However, there are significant motivating reasons to still aim at a scalable implementation for greedy hill climbing ("HC"). Besides providing an approximation guarantee, this algorithm has the added advantage of being able to generalize to a broader range of diffusion models (such as SEIR [20], SEIRS [21], and SEAIR [22]) that are used in epidemic simulations. The algorithm is also better equipped to incorporations of constraints during selection (e.g., fairness) that become important for epidemic simulations [19]. From an algorithmic standpoint, the HC algorithm also has the desired property of being able to subselect seed sets of smaller sizes, without having to recompute [18], in contrast to an approach like IMM which computes solutions for a pre-specified seed set size. The requirement for dynamically varying seed set sizes (incrementally) can emerge in the context of vaccination, due to manufacturing uncertainties and availability, as well as potential fairness constraints.

IV. Distributed CPU-GPU Implementation

In what follows, we present PREEMPT-HC—our parallel algorithm based on greedy hill climbing, to exploit the submodular property of PREEMPT. Our algorithm is a CPU-GPU algorithm that targets heterogeneous clusters.

Algorithm 2 shows the main steps of the parallel algorithm. It proceeds in three steps: sampling, counting, and seed selection. The Sampling step computes SG, a set of η subgraphs (random samples) of G obtained by probabilistically removing edges as per the targeted diffusion process. In the case of the SIR model, edges (i, j) will be removed with probability $1 - \omega_{ij}$. After sampling, the counting step proceeds iteratively by computing the expected gain in spread that can be achieved by the addition of any node $v \in V \setminus S$, over the collection of samples in SG. In the SIR model, this quantity can be computed as the number of visited nodes in a breadth-first search (BFS) from the vertices in S. Finally, S is greedily extended by including a vertex v that provides the maximum marginal gain in spread. The algorithm terminates when cardinality of S reaches k. This leads to the computational complexity of $O(k\eta|V|(|V| + |E|)$, where the cost of a BFS is given by $O(|V| + |E|)$. In an ideal parallel setting, the cost becomes $O((k\eta|V|(|V|+|E|)/p)$, by partitioning SG over p processes and ignoring communication costs or other overheads.

Algorithm 2: PREEMPT-HC: Selects a set of nodes S of size at most k that maximize the influence, $\sigma(\cdot)$.

```
Input  : $(G = (V, E, \omega)$, $k$, $\eta)$
Output: $S$
1 $SG \leftarrow$ Sampling ($G$, $\eta$)
2 $S \leftarrow \emptyset$
3 while $|S| \leq k$ do
4     $\forall_{v \in V}(count[v] = 0)$
5     for $v \in V \setminus S$ do in parallel
          // Counting Phase
6         for $G_i \in SG$ do
7             $count[v] \leftarrow \sigma_{G_i}(S \cup \{v\}) - \sigma_{G_i}(S)$
      // SeedSelect Phase
8     $s_{\text{best}} \leftarrow \arg\max_{v \in V}(count[v])$
9     $S \leftarrow S \cup \{s_{\text{best}}\}$
```

Implementation. At the high-level, implementing Algorithm 2 on a heterogeneous distributed system needs careful design consideration, as each processing element requires different optimizations and, sometimes, different algorithmic approaches to provide optimal performance. Code running on multicore CPUs needs to exploit task parallelism; especially for an algorithm like PREEMPT-HC which uses a large number of BFS passes on different samples of the graph, it also needs to account for reduced effectiveness of the cache hierarchy. Code running on GPUs needs to exploit data parallelism, reducing as much as possible branch divergence in vector-like computation, and align/coalesce memory accesses to exploit the massive memory bandwidth. Heterogeneous implementations also need to efficiently exploit the bandwidth between the different types of processing element, which is significantly lower than the bandwidth between a processing element and its own memory. This is achieved generally by minimizing the amount of data moved, but also by making sure that, if there is frequent communication, each transaction packs enough data to account for the transfer overheads. Because CPU and GPU tasks execute at different speeds and with different data volumes, load balancing is also required. The same communication challenges also arise when parallelizing across the nodes of the distributed system. PREEMPT-HC starts by storing the input graph G (replicated across nodes) in the CPU memory in compressed sparse row format (CSR).

Sampling: The sampling step of Algorithm 2 (Line 1) produces a collection of subgraphs from the input graph G. There are two possible approaches: explicitly storing each subgraph as a new graph data structure or implicitly storing it through a bit mask that disables the inactive edges. While explicitly storing the graph has the advantage that the counting step will only access active edges, it is more storage expensive and can become a significant cause for data movement between the host and the device memory. In our implementation, we choose to implicitly store the subgraphs using an edge-wise bit mask (see §VI-B for more details on the memory footprint).

Counting: The counting step of Algorithm 2 (Line 5) is implemented with a custom dynamic scheduling engine to support our hybrid implementation. Upon creation, the engine will instantiate a number of CPU and GPU worker threads according to the configuration requested by the user. The GPU workers will build a dedicated CSR graph data structure, inside their respective local device memory. There can be multiple GPU worker threads for a single GPU device. *We found that implementing the counting step on each CPU worker as a sequential BFS, and on each GPU worker as a parallel BFS provides the best performance for the SIR model.* This allows exploiting task parallelism with CPU cores, and data parallelism with the GPUs. Our implementation of the GPU workers use a modified version of the state-of-the-art parallel BFS provided in nvGRAPH[28]. This allows accounting, for a good extent, of branch divergence and memory coalescence. Our modifications include: allowing the BFS to start from multiple sources (by providing a frontier rather than a single source node), changing the edge mask into a bit mask to minimize data movement from the host, and adding the pruning strategy presented in PREEMPT.

As we will see in the results (Section VI), computing $\sigma(.)$ for the counting step is the most expensive step of computation. We highlight that recomputing the reachability information $(\sigma_{G_i}(S \cup \{v\}))$ from scratch is highly inefficient. In fact, the reachability information of what is already covered by S $(\sigma_{G_i}(S))$ would be computed $O(\eta|V \setminus S|)$ times. Our implementation avoids this duplicated work by caching visited nodes from S over the collection SG. The cache is later used to prune work in the BFS kernels so that they explore the portion of G_i that is contributed by the addition of v.

Dynamic scheduling: The cache building process and the

Sampling step also take advantage of the custom dynamic scheduling engine. Our dynamic scheduler serves two purposes: load balancing across the CPU cores and GPU devices, and pushing enough work to the GPUs to keep them fully utilized. The scheduler leverages a work queue (optimized using a single atomic counter) shared among all workers. GPU workers perform non-blocking calls to the GPU through streams to overlap computation with communication. The latest GPU architectures, such as Volta, allow parallel execution of kernels in different streams (HYPER-Q) if enough resources are available. Section VI evaluates the effectiveness of our dynamic scheduling approach.

Seed selection: The seed selection step of Algorithm 2 (Line 8) is one of finding a vertex v that provides the maximum marginal increment to the rechability of S. In distributed settings, this step requires a collective operation. In our performance study, we tried two different approaches: one based on one-sided communication with *count* distributed across the nodes, and the other using an `AllReduce`. Both methods yield a net communication cost of $O(k|V|\lg p)$, where p is the number of allocated ranks. At the scale of our experiments, we did *not* observe any significant difference in performance and therefore we have decided to leave the choice to the user to enable one or the other.

Algorithm 3: PREEMPT-RR: Selects a set of nodes S of size at most k that maximize the influence, $\sigma(\cdot)$. The set of reverse reachable paths is denoted by $\underline{\mathcal{R}}$.

```
Input  : $(G = (V, E, \omega),\ k,\ \epsilon)$
Output: $S$
while $|S| \leq k$ do
    $\langle \underline{\mathcal{R}}, \theta \rangle \leftarrow$ EstimateTheta($G,\ k,\ \epsilon$)
    $\underline{\mathcal{R}} \leftarrow$ ExpandR($G,\ \theta - |\underline{\mathcal{R}}|,\ \underline{\mathcal{R}}$)
    $S \leftarrow$ CountAndSelectSeeds($G,\ k,\ \underline{\mathcal{R}}$)
return $S$
```

Reverse Reachable Algorithm. While PREEMPT-HC can be generalized to several diffusion models, its cost can be a limiting factor for several application. The algorithm of Tang *et al.* [27], shown in Algorithm 3, provides excellent speedup when the diffusion model is SIR. Intuitively, a *reverse reachable* method works by asking the reverse question for a given vertex: Who can activate me? Tang *et al.* improve the original work of Borgs *et al.* [29] with a better bound on the number of reverse reachable sets computed, and consequently the amount of work done, by exploiting the statistical properties of the problem. This is shown as the `EstimateTheta` function in Algorithm 3, which includes both an `ExpandR` and a `CountAndSelectSeeds` step in a martingale loop. The `ExpandR` step now includes sampling of edges as well as the computation of reverse reachability information from randomly selected targets. The `CountAndSelectSeeds` picks k seeds from V by solving a maximum coverage problem over the collection or reverse reachable sets ($\underline{\mathcal{R}}$). Minutoli *et al.* [30] recently developed a distributed, as well as a distributed multi-GPU, implementation of [27]. Algorithm 3 runs in $O((k + l)(|V| + |E|)\log|V|/\epsilon^2)$ and returns a $(1 - 1/e - \epsilon)$-approximation, for some $\epsilon > 0$, with probability $1 - 1/|V|^l$. The communication cost of a parallel implementation is $O(k|V|\lg p)$, where p is the number of processes. We provide scaling results of Algorithm PREEMPT-RR in §VI-B for comparison with PREEMPT-HC.

Software Implementation and Availability. We implemented PREEMPT-RR and PREEMPT-HC using C++ and the MPI, OpenMP, and CUDA programming models. Their source code is distributed as part of Ripples[31].

V. Experimental Setup

Testbed: Distributed Multi-GPU Supercomputer. We execute our implementation on Summit, a US Department of Energy's Leadership-class machine hosted at Oak Ridge National Laboratory [32]. As of this writing, Summit is ranked number 1 in the TOP500 list[33]. The system consists of $4,608$ nodes, each equipped with two POWER9 CPUs, 6 NVIDIA Tesla V100 GPUs, 512 GB of DDR4 RAM, and two Infiniband EDR network interfaces. A POWER9 CPU hosts 22 cores (1 is reserved for system software) with 4 threads each, has a frequency of 3.07 GHz and hosts 110 MB of L3 cache. Each Volta GPU hosts 80 Streaming Multiprocessors (64 FP32, 64 INT, 32 FP64 and 8 Tensor Cores running at 1.333 GHz) with 16 GB of HBM2 memory (1750 Mhz on a 4096-bit bus, providing 900 GB/s of bandwidth). Group of 3 GPUs are directly connected with a POWER9 CPU and among each other with NVLINK2 connections. Each connection uses 2 of the 6 NVLINK2 channels of a Volta GPU, offering up to 100 GB/s of bidirectional bandwidth. A group, composed of 3 GPUs and a CPU, communicates with the other one through the X-Bus between the two CPUs, with a maximum bandwidth of 64 GB/s. Our code was compiled with gcc 8.1.1 and CUDA 10.1.243; we use the spectrum MPI library 10.3.1.2.

TABLE II: SNAP (top) and Contact Networks (bottom)

Graph	Nodes	Edges	Avg. Degree	Max Degree
HepPh	34,546	421,578	24.41	846
Slashdot	77,360	905,468	23.41	5,048
Epinions	75,879	508,837	13.41	3,079
DBLP	317,080	1,049,866	6.62	343
Google	875,713	5,105,039	11.66	6,353
BerkStan	685,230	7,600,595	22.18	84,290
LiveJournal	4,847,571	68,993,773	28.47	22,889
Orkut	3,072,441	117,185,083	76.28	33,313
Portland	1,501,209	21,155,681	13.78	487
Portland-141k	63,543	141,450	2.23	25
Portland-295k	131,624	295,281	2.24	25
Portland-415k	182,967	415,434	2.27	26

Input Datasets. We use two types of datasets for empirical evaluations. The first type of networks, listed in Table II, constitutes social contact graphs for Portland, OR, and Montgomery county, VA, built using agent-based models and a first principles approach [15], [16]. Such networks are commonly used in public health analyses, since they capture several attributes of the underlying population [2], [34]–[38]. The other type of networks, listed in Table II, are from the SNAP collection [39]. These inputs originate from a variety of application contexts and therefore provide a means for performance expectation for submodular optimization that goes beyond the epidemiology use-case. The weights on the edges of these inputs were randomly generated from a uniform distribution between $[0; 1]$. Ripples [31] and our experimental setup [40] contain all the necessary information to reproduce the results presented in this paper.

VI. Experimental Results

We first present a thorough qualitative evaluation of PREEMPT by examining various output attributes and comparing them with the mathematical programming based approach of Sambaturu *et al.* [12]. We then provide scaling results for PREEMPT on Summit. All results for PREEMPT were generated using PREEMPT-HC, unless otherwise stated.

A. *Qualitative Evaluation of* PREEMPT

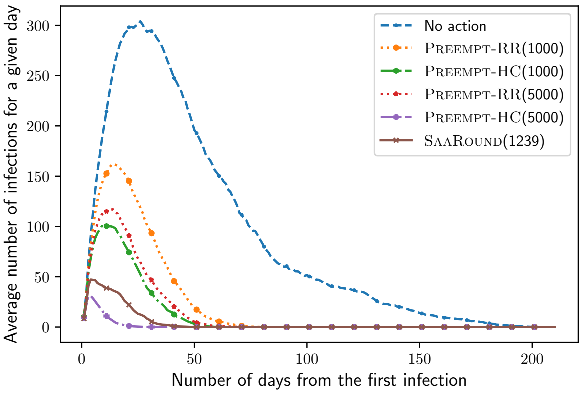


Fig. 3: Time series of infections ("epicurve") comparison for PREEMPT-HC, PREEMPT-RR and SAAROUND[12] for Portland-141k. We note that the peak moves earlier in the process, but optimization for the peak is a different problem.

We use the solution computed by PREEMPT as the intervention in an SIR simulation on the contact networks, and determine the number of infections in the simulation. Better qualitative performance corresponds to lesser number of infections (i.e., more lives saved). For the SIR simulation, we assume that the disease starts at a small set (size≈10) of random initial nodes, and a uniform transmission probability $w_{uv}{=}p$ for each edge (u, v) is chosen so that in the no-action case (i.e., WITHOUT PREEMPT), the expected number of infections is at least 25% of the network. With PREEMPT, a vaccination budget (k) is used. Fig. 1 shows that even with relatively low budgets of 1000 and 5000 (0.4% and 2% of the population resp.), PREEMPT is able to achieve significant reductions over WITHOUT PREEMPT, in the number of infections—between factors of 2.33× (for Portland-295k) to 6.75× (Portland-141k).

Fig. 3 shows the effectiveness of the different schemes PREEMPT-HC, PREEMPT-RR, SAAROUND[12] compared to the baseline which represents no-action/intervention. The curves suggest a clear value in intervention, yielding between $2\times$ and $9\times$ reductions in the number of infections. Also, it can be seen that PREEMPT-HC outperforms PREEMPT-RR by reducing the number of infections significantly—for instance, on a budget of 5000, the *total* number of infections can be decreased by 98.57% for PREEMPT-HC, compared to the no-action baseline; whereas the decrease is 83.89% for PREEMPT-RR. As expected PREEMPT-RR is significantly faster than PREEMPT-HC (see §VI-B) and SAAROUND. Furthermore, the performance of PREEMPT-HC is comparable to SAAROUND, both in terms of quality and runtime (3-5 hours) for a moderately sized input such as Portland-141K—for instance, the total number of infections under PREEMPT-HC(5000) was 271, compared to the 869 infections under SAAROUND(1239). Note that we were not able to run SAAROUND for the larger inputs with millions of nodes because of time limitations. Fig. 3 also shows the temporal value of intervention (via vaccination) as the peak of infections clearly move left (i.e., earlier in time) compared to the curve with no intervention. The peak height and the timing of the peak, can profoundly impact treatment as both are linked to hospitalization and hospital capacities.

Demographic characteristics of PREEMPT. Fig. 4 shows the age and degree of the nodes picked in the solution, and their correlation. We note that there is a significant spread in the ages of the nodes selected. This spread is interesting because it suggests that the common public health strategies that tend to focus on specific age groups (particularly children) may not be the most effective for EPICONTROL.

Node characteristics of PREEMPT. As mentioned in §VII, a number of network properties have been used for selecting nodes to intervene, including degree, clustering coefficient,

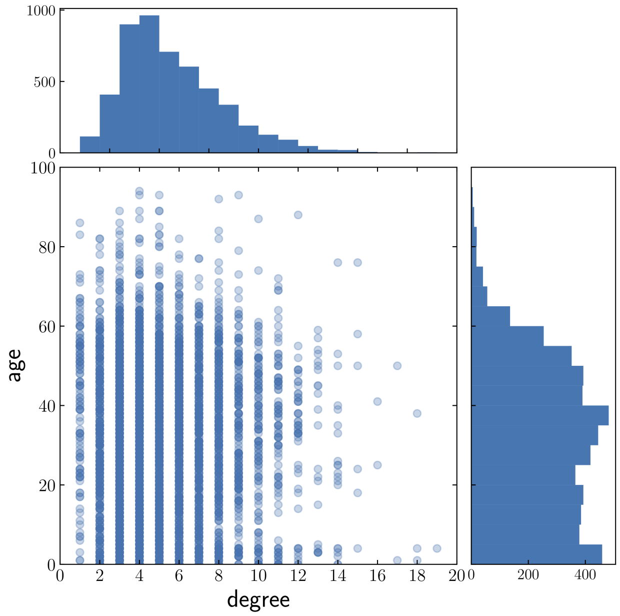


Fig. 4: *Degree vs. Age* for the seeds selected by PREEMPT on Portland_415k with their degree (top) and age (right) distribution.

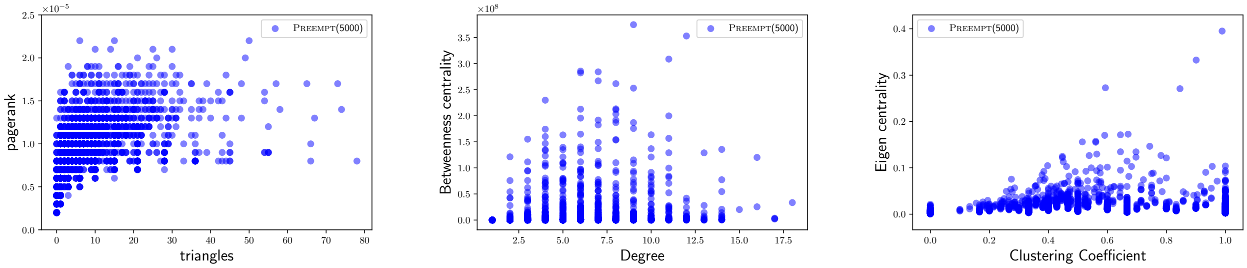


Fig. 5: *Correlations* between different measures of graph structural characteristics of the nodes picked by PREEMPT: (Left) #Triangles vs. Pagerank; (Middle) Degree vs. Betweenness Centrality; and (Right) Clustering Coefficient vs. Eigenvector Centrality.

and various notions of centrality. Fig. 5 shows different node characteristics in the solutions computed by PREEMPT, and correlations between them. The results show that there is no single node characteristic that could be used as a "surrogate" to design good interventions in synthetic social network models used in public health. In particular, there is a wide spread in both the degree and clustering coefficient (Fig. 5, middle and right). This contrasts with the performance of the degree heuristic for scale free networks [7], [41]. Some of the common centrality measures—pagerank, betweenness centrality, and eigen centrality, also do not show a significant dominance; among these, the pagerank of the selected nodes seems to be somewhat high, and that could serve as a promising surrogate, though even that has a fairly large spread.

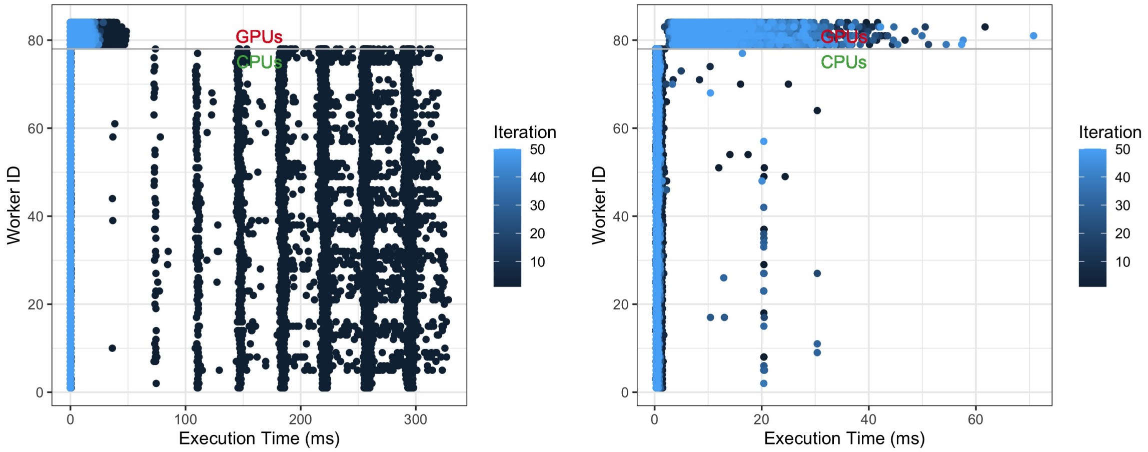
B. Performance Evaluation on Summit

We present performance results and evaluation for executing PREEMPT-HC on Summit. We also present strong scaling results on Summit for PREEMPT-RR.

Single Node Performance of PREEMPT-HC. In order to understand the performance on a single node, we study the behavior of task execution on CPU versus GPU as the algorithm progresses reflecting varying degrees of computational load. We measured the execution time of the first 100 tasks executed by 84 workers (6 GPU workers and 78 CPU workers) for each of the 50 iterations used to select 50 seeds (one per iteration). The results are presented in Fig. 6. We observe that tasks from early iterations (dark blue color) of the algorithm are relatively more expensive than tasks from later iterations (light blue color). The runtime behaviour reflects the effectiveness of the pruning strategy that we incorporated in PREEMPT (as described in §IV).

We further observe that the inputs from the SNAP collection and the two Contact Networks have contrasting execution profiles. We show one representative instance from each in Fig. 6—DBLP on the left and Portland on the right. Tasks for the Contact Networks run very quick and the overheads of offloading the computation to the accelerator negate the potential benefits. The total number of tasks is given by $(\eta \cdot |V|)/p$, where p is the number of processing units (threads). In contrast, the inputs from the SNAP collection have the opposite behavior as the early iterations (dark blue) run considerably faster on the GPUs than on the CPUs (on average on DBLP first iteration: 6.91×). However, these gains on GPUs are reversed during the final iterations of the algorithm were CPUs become faster than GPUs. In fact during the last iteration on DBLP, CPU tasks execute on average in $0.03ms$ while GPU tasks execute on average in $8.6ms$. The loss of efficiency for GPUs can be attributed to the overhead of offloading the computation to GPUs when not enough computation remains. Thus, dynamic task scheduling between CPU and GPU workers becomes an important strategy to adapt to variations in computational load and data movement.

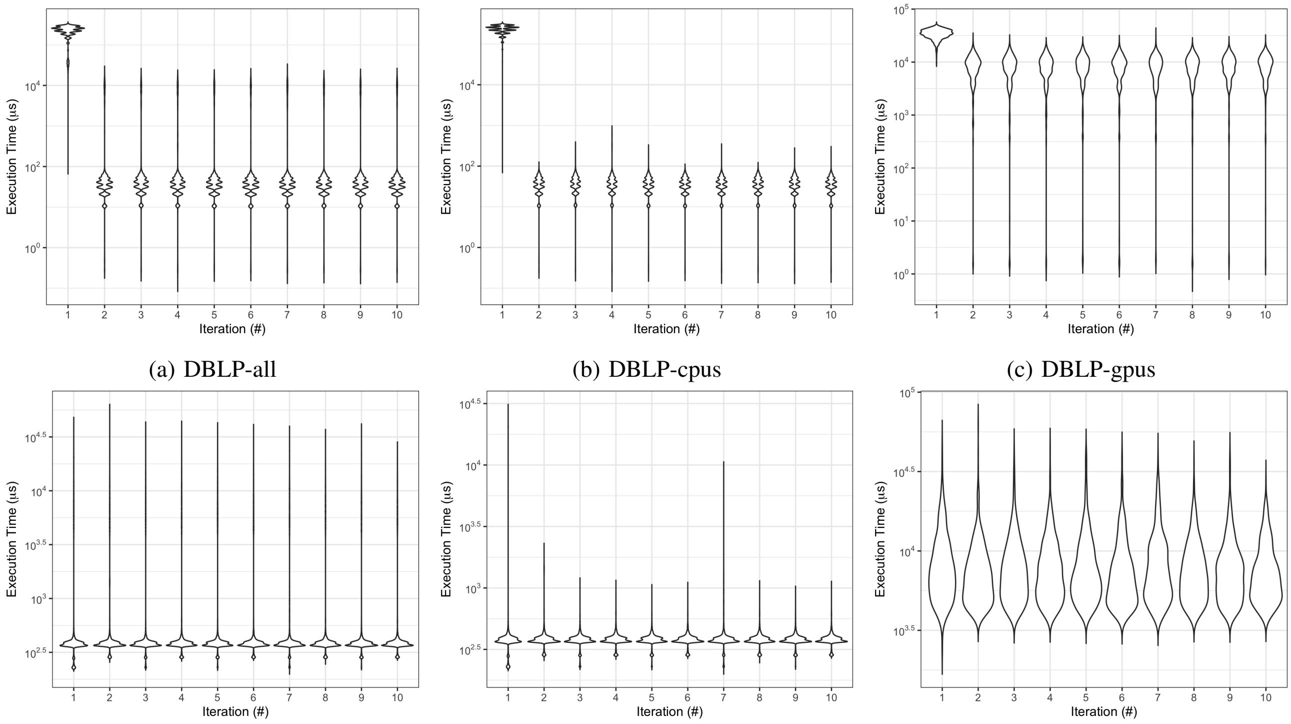
In order to provide a better insight on the execution behavior, we present a portion of the data using violin plots [42]. We opted for the violin plots due to their versatility in capturing variability in data, not only from the execution times on CPUs versus GPUs, but also across different measurements on CPUs and GPUs by themselves, as well as the differences between different inputs. Ridges in the figure show multimodal distribution, and the long tails characteristic of lognormal distribution show the skewedness of the execution time distribution (we note that a normal distribution produces a smooth violin plot). In Fig. 7, we show the distribution of the execution times of the tasks obtained by profiling the counting phase in Algorithm 2 during the first 10 iterations of the algorithm. Measurements were taken with 84 total threads (78 CPU workers and 6 GPU workers). We observe a more skewed distribution for the Portland dataset than for the DBLP dataset (Fig. 7a and 7d), and up to four orders of magnitude difference between the fastest and the slowest tasks in each iteration (long tail). Similar observations emerge also from analyzing CPU and GPU tasks in isolation (Fig. 7b, 7c, 7e and 7f). DBLP shows a clear shift of regimen between the first iteration and the subsequent iterations, which is caused by the pruning strategy included in PREEMPT-HC (which leads to potential reduction in work where applicable). In fact, while the pruning strategy is shifting the distribution towards faster execution times, there is still great variability in runtime time that can be attributed directly to the structural and numerical properties of a given input. For Portland, we do not observe a similar reduction after the first iteration. We can in fact observe a scenario where the pruning strategy is less effective when the edge probabilities are considerably low; peculiar to this dataset. We conclude that dynamical scheduling of tasks on computing resources is not



(a) DBLP

(b) Portland

Fig. 6: PREEMPT-HC *Task Execution Time* for the first 100 tasks executed by 84 worker threads (Rank 0 to Rank 77 on CPUs on the bottom; Rank 78 to Rank 83 on GPUs on the top) for 50 iterations of the algorithm, for two inputs (DBLP on the left and Portland on the right) on Node Zero of Summit. Each iteration selects one seed. There is a significant difference in the performance behavior for the two inputs, and in the performance on CPUs versus GPUs for early versus later iterations.



(a) DBLP-all

(b) DBLP-cpus

(c) DBLP-gpus

(d) Portland-all

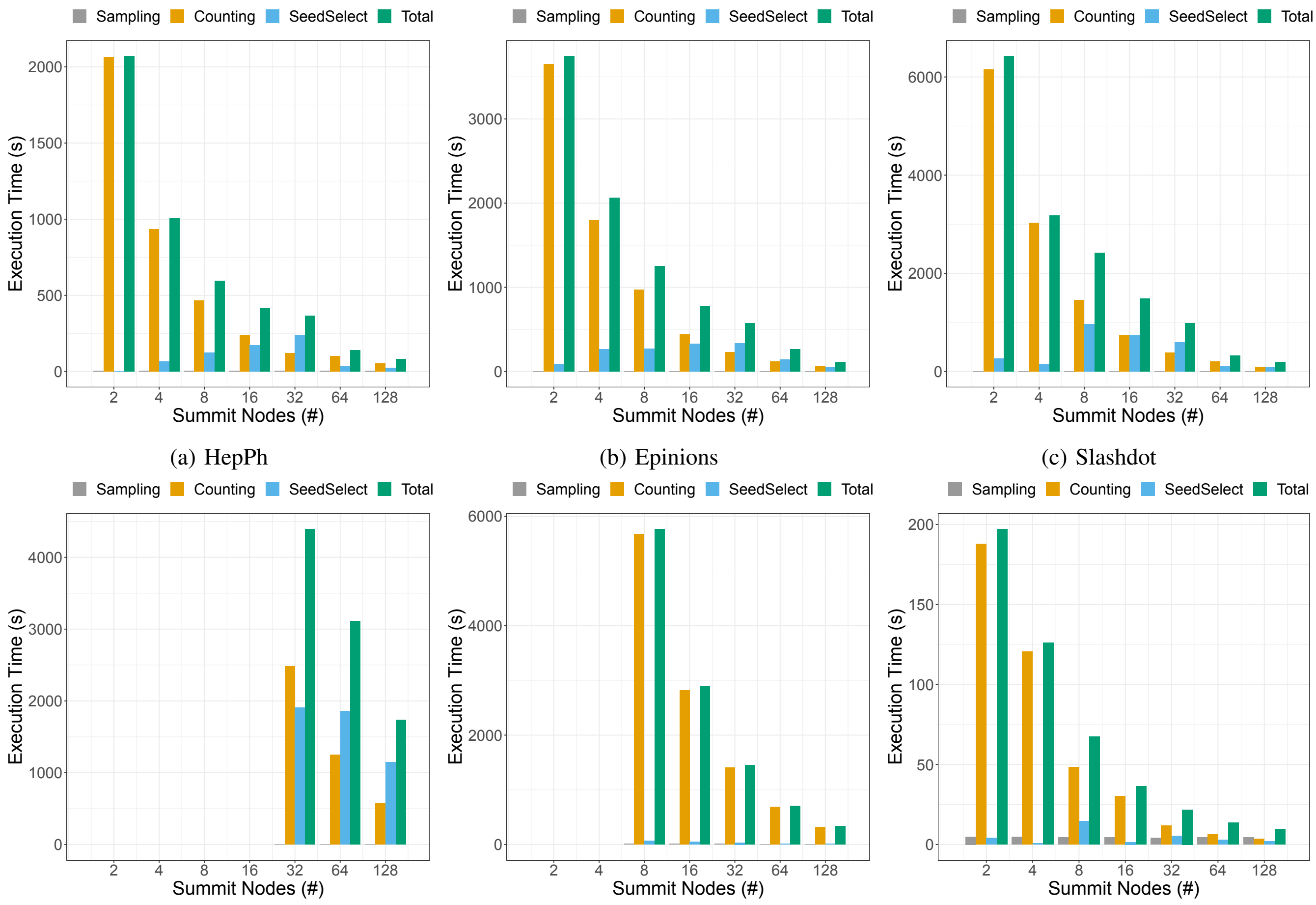
(e) Portland-cpus

(f) Portland-gpus

Fig. 7: An illustration of execution time variability using violin plots for PREEMPT-HC. The plot highlights a portion of the data presented in Section VI-B for the first 100 tasks executed by 84 worker threads (78 on CPUs and 6 on GPUs). Top row is DBLP and the bottom row is Portland, on Node Zero of Summit. We report the distribution for all tasks aggregated together (left column), and a detailed view considering only CPU tasks (center column) and GPU tasks (right column) separately. Iteration number (from one to ten) is plotted along the **X-axis**, and execution time (in log scale) is plotted along the **Y-axis**.

only important to address when there is a difference between processing elements on heterogeneous platforms (e.g., CPUs and GPUs), but is also fundamental to adapt to scenarios where the execution time is critically dependent on the data, such as the execution of PREEMPT-HC.

Strong Scaling of PREEMPT-HC. We present scaling results with up to 128 nodes on Summit that amount to a total of 5,376 CPU cores and 768 GPUs. For the experiments we set



(a) HepPh

(b) Epinions

(c) Slashdot

(d) DBLP

(e) Portland

(f) Montgomery

Fig. 8: *Strong Scaling* for PREEMPT-HC on the SNAP networks (a-d) and the Contact Networks (e-f). Please note that the missing data points are executions that did not complete under two hours.

the number of samples (η) to 1000 and number of seeds (k) to 50. Fig. 8 show the results for both the SNAP collection and Contact Networks. We report the total execution time including the break down of the three phases for the entire execution with 50 iterations. We observe that the Counting step dominates the runtime. There are few exceptions where the cost of seed selection becomes larger. The variability in seed selection's performance is caused by the network communication (`AllReduce`) that is required during the step. However, on production runs where the number of samples (η) used will be set beyond 10000, the impact of this variation in performance of seed selection diminishes because the computation is likely to be dominated by the Counting step (§IV) that scales with large speedups (e.g., Slashdot: 63×, Montgomery: 155×). Overall, we observe that our implementation is able to scale well on up to 128 Summit nodes, on both data sets, generating speedups between 20× and 33× relative to the configuration with two nodes of Summit. Notably, we are able to significantly reduce time to solution from hours to minutes (e.g., for Slashdot it takes 6,424 seconds on two Summit nodes vs. 192 seconds on 128 Summit nodes).

Strong scaling of PREEMPT-RR. Due to recent advances in algorithms, PREEMPT-RR provides the best performance both in serial and in parallel [27], [30]. However, the idea of reverse reachability is not easily generalized to different models of diffusion that are needed in practice, but will be beneficial for simpler models such as SIR. In order to provide a relative comparison with PREEMPT-HC, we provide results from the execution of CURIPPLES from Minutoli *et al.* on Summit; illustrated in Fig. 9. We configured the parameters of CURIPPLES to provide a $1/2$-approximate solution ($\epsilon = 0.13$), and therefore representative of what to expect in production runs. We note that LiveJournal has a missing data point for the configuration with two nodes since the aggregated 1TB of memory was not sufficient to store the intermediate information generated. Our results demonstrates that CURIPPLES scales considerably well up to 128 nodes on Summit, with speedups between 2× and 25× relative to the performance using two nodes. CURIPPLES is also considerably faster than PREEMPT-HC. This is to be expected as greedy hill climbing is computationally more expensive than the reverse reachable approach implemented in CURIPPLES. However, PREEMPT-HC has two advantages over CURIPPLES: it is more memory-efficient (as described below), and it is more generic and extensible to other models (as described in §III-C).

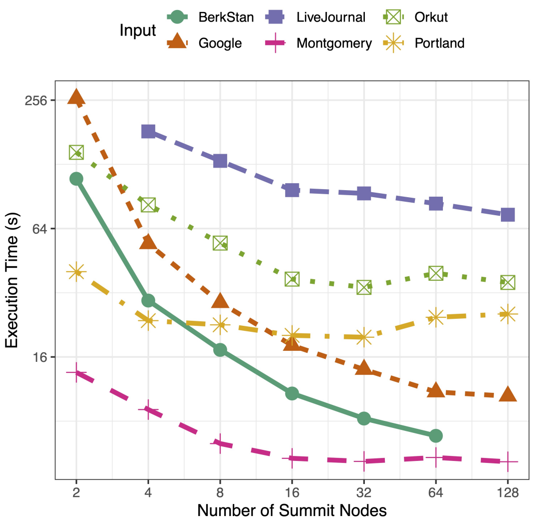


Fig. 9: *Strong Scaling* for CuRipples on the SNAP networks. We are investigating the BerkStan run on 128 nodes.

Memory Footprints. Memory requirements for PREEMPT-RR is a function of the input graph characteristics such as connectivity and edge probabilities, as well as the input parameters that specify accuracy (ϵ) and number of seeds (k). In particular, memory footprint depends on the number of reverse reachable sets ($\mathcal{R}$) that need to be computed, which grows nonlinearly with respect to k and ϵ. We refer you to Minutoli *et al.* [43] for further details.

In contrast, the memory footprint of PREEMPT-HC is predictable from the input parameters. PREEMPT-HC needs to store η subgraphs sampled from the input. Each subgraph can be compactly stored using a bitmap that enables/disables edges in a given sample instead of explicitly storing them as real numbers or by creating separate adjacency lists. In such implementation, the storage (in bits) required for the sampled graphs is bounded $\Theta(\eta|E|)$, where η and E are both known a priori. Minutoli *et al.* [43] report that the reverse reachability method requires more that $3TB$ of memory to solve an instance of the problem on the Orkut graph ($\epsilon = 0.13$, $k = 200$). PREEMPT-HC on same instance using 10000 samples (standard settings in the literature) will require only about $146GB$.

VII. Related Work

In the traditionally studied area of ordinary differential equations (ODE) based epidemic models, it is possible to find optimal solutions using brute force search methods, e.g., [1]. However, designing effective interventions for network (graph theoretic) based models is a much harder problem. The methods under these discrete models can be broadly classified into the following three categories:

(1) *Interventions based on static network structure*: These involve picking nodes in rank order of properties, such as degree, pagerank, and centrality, e.g., [5]–[7]. In many power law network models, degree based strategies have been shown to work very well empirically [7], and with rigorous guarantees [41]. Centrality based strategies are computationally much more expensive, and harder to analyze.

(2) *Algorithms using spectral properties*: there are interesting connections between epidemic dynamics and spectral properties, namely the spectral radius and the spectral gap [44]–[46]. There has been a lot of work on reducing the spectral radius [8]–[11]. However, these algorithms do not scale very well to large networks.

(3) *Algorithms based on math programming*: this line of work has involved using linear relaxations of integer programming formulations, and then rounding them to get near-optimal solutions with rigorous guarantees [12]–[14]. Only the work of [12] presents empirical evaluation, but these methods do not, in general, scale to very large networks.

The problem of influence maximization is a closely related topic that can be seen as the inverse of epidemic containment. Beginning with the seminal work of Domingos and Richardson [47] and Kempe *et al.* [17], it has been studied extensively. The key contribution in these works is the demonstration that the influence function is submodular, meaning that it can be maximized by greedy approximation algorithms [48]. An extensive body of recent work has focused on reducing the super quadratic time complexity through a variety of statistical and data structure techniques [29], [49], [50]. While many works look at optimizing and applying the serial influence maximization algorithms to diverse domains, there are only a few very recent works on parallelizing these algorithms. [30], [43], [51]. Particularly relevant for our work is the implementation of the IMM algorithm on multi-GPU systems presented in [30]. However, given the limited applicability of IMM, in this work we consider the parallelization of the Greedy Hill Climbing algorithm. To the best of our knowledge, our effort is the first to scale this algorithm on modern clusters with multiple GPUs per node.

VIII. Conclusion and Future Work

Under the context of epidemic control of a disease, we presented an efficient distributed algorithm based on influence maximization targeting state-of-the-art heterogeneous CPU-GPU supercomputers. The key contributions of this work are: i) Efficient implementation of the greedy hill climbing algorithm through a combination of custom kernels for each processing element; scheduling techniques to balance the workload between CPUs and GPUs varying amounts of computational loads; improved synchronization and reduction of data movement overheads. ii) A novel formulation for epidemic control when the initial set of infected nodes are not known through submodular optimization, and the proofs of submodularity under special cases. The formulations and parallel implementations show that we can successfully compute solutions at scale on Summit for large real-world inputs. To the best of our knowledge, this is the first distributed hybrid CPU-(multi)GPU implementation for greedy hill climbing algorithm and its application for epidemic control. We therefore believe that this work will benefit the epidemiology researchers and practitioners, as well as impact the future research in influence maximization and porting of applications to extreme scale architectures.

Future research directions include: i) adapting the influence maximization based approach for efficient testing for an epidemic outbreak, and for designing efficient vaccination under fairness constraints; ii) extensive testing of the current methods on different datasets including data specific for COVID-19 outbreak; iii) implementation of different diffusion models such as SI and SEIR to address novel outbreaks; iv) modifying the breadth-first search kernel to enable efficient execution of shorter searches under specific conditions; v) perform empirical analysis on carefully chosen inputs to identify realistic values for the number of samples (η); and, vi) performing larger production runs for specific models of contact networks to assist in decision making under uncertainties.

Acknowledgment

We used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory (Contract No. DE-AC05-00OR22725). The research is in parts supported by the U.S. DOE ExaGraph project at the Pacific Northwest National Laboratory (PNNL), by the U.S. National Science Foundation (NSF) grants CCF 1815467, OAC 1910213, and CCF 1919122 to Washington State University, and by National Institutes of Health (NIH) Grant 2R01GM109718-07 and NSF grant numbers IIS-1955797, IIS-1633028, CCF-1918656, and DTRA subcontract/ARA S-D00189-15-TO-01-UVA, and a collaborative seed grant from the UVA Global Infectious Disease Institute. PNNL is operated by Battelle Memorial Institute under Contract DE-AC06-76RL01830. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

[1] J. Medlock and A. P. Galvani, “Optimizing influenza vaccine distribution,” *Science*, vol. 325, no. 5948, pp. 1705–1708, 2009.

[2] M. E. Halloran *et al.*, “Modeling targeted layered containment of an influenza pandemic in the United States,” in *Proceedings of the National Academy of Sciences (PNAS)*, 2008, pp. 4639–4644.

[3] E. Lofgren *et al.*, “Opinion: Mathematical models: A key tool for outbreak response,” *Proceedings of the National Academy of Sciences (PNAS)*, pp. 18 095–18 096, 2014.

[4] S. Finbow and G. MacGillivray, “The firefighter problem: A survey of results, directions and questions.,” *Australasian J. Combinatorics*, vol. 43, pp. 57–78, 2009.

[5] R. Cohen *et al.*, “Efficient immunization strategies for computer networks and populations,” *Phys. Rev. Lett.*, vol. 91, p. 247 901, 24 2003. DOI: 10.1103/PhysRevLett.91.247901.

[6] J. C. Miller and J. M. Hyman, “Effective vaccination strategies for realistic social networks,” *Physica A: Statistical Mechanics and its Applications*, pp. 780–785, 2007.

[7] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, 1999. DOI: 10.1126/science.286.5439.509.

[8] V. M. Preciado *et al.*, “A convex framework to control spreading processes in directed networks.,” in *Annual Conference on Information Sciences and Systems (CISS)*, IEEE, 2014.

[9] V. M. Preciado *et al.*, “Optimal vaccine allocation to control epidemic outbreaks in arbitrary networks.,” in *IEEE Conference on Decision and Control*, IEEE, 2013.

[10] S. Saha *et al.*, “Approximation algorithms for reducing the spectral radius to control epidemic spread,” in *SIAM SDM*, 2015.

[11] Y. Zhang *et al.*, “Controlling propagation at group scale on networks,” in *Data Mining (ICDM), 2015 IEEE International Conference on*, IEEE, 2015, pp. 619–628.

[12] P. Sambaturu *et al.*, “Designing effective and practical interventions to contain epidemics,” in *Proc. AAMAS*, 2020.

[13] J. Aspnes *et al.*, “Inoculation strategies for victims of viruses and the sum-of-squares partition problem,” in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA ’05, Vancouver, British Columbia, 2005, pp. 43–52, ISBN: 0-89871-585-7.

[14] A. Hayrapetyan *et al.*, “Unbalanced graph cuts,” in *ESA*, 2005, pp. 191–202.

[15] S. Eubank *et al.*, “Modelling disease outbreaks in realistic urban social networks,” *Nature*, vol. 429, pp. 180–184, 6988 2004.

[16] C. L. Barrett *et al.*, “Generation and analysis of large synthetic social contact networks,” in *Winter Simulation Conference*, Winter Simulation Conference, 2009, pp. 1003–1014.

[17] D. Kempe *et al.*, “Maximizing the Spread of Influence Through a Social Network,” in *Proceedings of ACM SIGKDD*, Washington, D.C.: ACM, 2003, pp. 137–146. DOI: 10.1145/956750.956769.

[18] E. Cohen *et al.*, “Sketch-based influence maximization and computation: Scaling up with guarantees,” in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014, pp. 629–638.

[19] A. Tsang *et al.*, “Group-Fairness in Influence Maximization,” in *ArXiv*, vol. abs/1903.00967, 2019.

[20] J. C. Blackwood and L. M. Childs, “An introduction to compartmental modeling for the budding infectious disease modeler,” *Letters in Biomathematics*, vol. 5, no. 1, pp. 195–221, 2018. DOI: 10.1080/23737867.2018.1509026.

[21] L. R. Lopez and X. Rodo, “A modified SEIR model to predict the COVID-19 outbreak in Spain and Italy:

simulating control scenarios and multi-scale epidemics," *medRxiv*, 2020. DOI: 10.1101/2020.03.27.20045005.

[22] R. N. Thompson *et al.*, "Detecting Presymptomatic Infection Is Necessary to Forecast Major Epidemics in the Earliest Stages of Infectious Disease Outbreaks," *PLoS Comput. Biol.*, vol. 12, no. 4, 2016. DOI: doi: 10.1371/journal.pcbi.1004836.

[23] M. Marathe and A. Vullikanti, "Computational Epidemiology," *Communications of the ACM*, vol. 56, no. 7, pp. 88–96, 2013.

[24] R. Anderson and R. May, *Infectious Diseases of Humans*. Oxford: Oxford University Press, 1991.

[25] M. L. Fisher *et al.*, "An analysis of approximations for maximizing submodular set functions—ii," in *Polyhedral Combinatorics: Dedicated to the memory of D.R. Fulkerson*, M. L. Balinski and A. J. Hoffman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 73–87. DOI: 10.1007/BFb0121195.

[26] G. L. Nemhauser *et al.*, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[27] Y. Tang *et al.*, "Influence Maximization in Near-Linear Time: A Martingale Approach," in *Proc. 2015 ACM SIGMOD International Conference on Management of Data*, ACM, 2015, pp. 1539–1554.

[28] NVIDIA, *Nvgraph*. [Online]. Available: https://developer.nvidia.com/nvgraph.

[29] C. Borgs *et al.*, "Maximizing social influence in nearly optimal time," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '14, Portland, Oregon: SIAM, 2014, pp. 946–957.

[30] M. Minutoli *et al.*, "cuRipples: Influence maximization on multi-GPU systems," in *Proceedings of the International Conference on Supercomputing 2020 (ICS20)*, 2020.

[31] M. Minutoli *et al.*, *Ripples: Source code*. DOI: 10.5281/zenodo.3942724. [Online]. Available: https://github.com/pnnl/ripples.

[32] S. S. Vazhkudai *et al.*, "The design, deployment, and evaluation of the coral pre-exascale systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18, Dallas, Texas: IEEE Press, 2018.

[33] *TOP500: The list*, 2020. [Online]. Available: https://www.top500.org/resources/top-systems/.

[34] T. C. Germann *et al.*, "Mitigation strategies for pandemic influenza in the united states," *Proceedings of the National Academy of Sciences*, vol. 103, no. 15, pp. 5935–5940, 2006. DOI: 10.1073/pnas.0601266103.

[35] M. F. C. Gomes *et al.*, "Assessing the international spreading risk associated with the 2014 West African Ebola outbreak," English, *PLoS Currents*, vol. 6, 2014, PMCID: PMC4169359. DOI: 10.1371/currents.outbreaks.cd818f63d40e24aef769dda7df9e0da5. [Online]. Available: http://currents.plos.org/outbreaks/?p=40803.

[36] N. Ferguson *et al.*, "Strategies for mitigating an influenza pandemic," *Nature*, vol. 442, pp. 448–452, Apr. 2006, PMID: 16642006.

[37] 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*, vol. 13, no. 1, p. 940, 2013, PMCID: PMC3852955.

[38] F. Liu *et al.*, "The role of vaccination coverage, individual behaviors, and the public health response in the control of measles epidemics: An agent-based simulation for california," *BMC Public Health*, vol. 15, no. 1, p. 447, 2015, PMCID: PMC4438575, ISSN: 1471-2458. DOI: 10.1186/s12889-015-1766-6. [Online]. Available: https://doi.org/10.1186/s12889-015-1766-6.

[39] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection*, http://snap.stanford.edu/data, Jun. 2014.

[40] M. Minutoli *et al.*, *PREEMPT: Experimental setup and script generator*. DOI: 10.5281/zenodo.3949293. [Online]. Available: https://github.com/mminutoli/preempt-experimental-setup.

[41] B. Bollobás and O. Riordan, "Robustness and vulnerability of scale-free random graphs," *Internet Mathematics*, vol. 1, pp. 1–35, 2003.

[42] J. L. Hintze and R. D. Nelson, "Violin Plots: A Box Plot-Density Trace Synergism," *The American Statistician*, vol. 52, no. 2, T. A. Statistician, Ed., pp. 181–184, 1998. DOI: 10.2307/2685478.

[43] M. Minutoli *et al.*, "Fast and Scalable Implementations of Influence Maximization Algorithms," in *2019 IEEE International Conference on Cluster Computing, CLUSTER 2019, Albuquerque, NM, USA, September 23-26, 2019*, IEEE, 2019, pp. 1–12. DOI: 10.1109/CLUSTER.2019.8890991.

[44] B. A. Prakash *et al.*, "Threshold conditions for arbitrary cascade models on arbitrary networks," in *ICDM*, 2011.

[45] A. Ganesh *et al.*, "The effect of network topology on the spread of epidemics," *Proceedings of INFOCOM*, 2005.

[46] Y. Wang *et al.*, "Epidemic spreading in real networks: An eigenvalue viewpoint," *Proceedings of SRDS*, 2003.

[47] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of ACM SIGKDD*, ACM, 2001, pp. 57–66.

[48] L. Wolsey, "An analysis of the greedy algorithm for the submodular set covering problem," *Combinatorica*, 385–393, 1982.

[49] W. Chen *et al.*, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proceedings of ACM SIGKDD*, ACM, 2010, pp. 1029–1038.

[50] J. Leskovec *et al.*, “Cost-effective outbreak detection in networks,” in *Proceedings of ACM SIGKDD*, ACM, 2007, pp. 420–429.

[51] M. Halappanavar *et al.*, “Accelerating the mining of influential nodes in complex networks through community detection,” in *Proceedings of the ACM International Conference on Computing Frontiers, CF’16, Como, Italy, May 16-19, 2016*, ACM, 2016, pp. 64–71.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We execute our implementation on Summit, a US Department of Energy's Leadership-class machine hosted at Oak Ridge National Laboratory. The system consists of 4,608 nodes, each equipped with two POWER9 CPUs, 6 NVIDIA Tesla V100 GPUs, 512 GB of DDR4 RAM, and two Infiniband EDR Network interfaces. A POWER9 CPU hosts 22 cores (1 is reserved for system software) with 4 threads each, has a frequency of 3.07 GHz and hosts 110 MB of L3 cache. Each Volta GPU hosts 80 Streaming Multiprocessors (64 FP32, 64 INT, 32 FP64 and 8 Tensor Cores running at 1.333 GHz) with a 16 GB of HBM2 memory (1750 MHz on a 4096-bit bus, providing 900 GB/s of bandwidth). Group of 3 GPUs are directly connected with a POWER9 CPU and among each other with NVLINK2 connections. Each connection uses 2 of the 6 NVLINK2 channels of a Volta GPU, offering up to 100 GB/s of bidirectional bandwidth. A group composed of 3 GPUs and a CPU communicates with the other one through the X-Bus between the two CPUs, providing a maximum bandwidth of 64 GB/s. Our code was compiled with gcc 8.1.1 and CUDA 10.1.243, and we used the spectrum MPI library 10.3.1.2.

We have performed scalability studies 2 to 128 nodes of the machine as detailed in the paper. PREEMPT-HC was configured to used 1000 samples and 50 seeds while cuRipples was configured to use epsilon=0.13 and k=100.

As part of the available artifacts, we have provided the Portland contact network used in the strong scaling study in Figures 6-8. The experimental setup repository contains all the data collected during our experiments and the needed scripts to regenerate the figures included in the paper.

Our experimental setup artifact also provides details about experimenting with other publicly available data sets. The tool will generate similar figures that can be used to validate the obtained results.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: There are no author-created data artifacts.

Proprietary Artifacts: No author-created artifacts are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: 10.5281/zenodo.3942724,
↪  https://github.com/pnnl/ripples
Artifact name: Ripples v2.0 (PREEMPT)

Persistent ID: 10.5281/zenodo.3949293, https://githu⌋
↪  b.com/mminutoli/preempt-experimental-setup
Artifact name: PREEMPT Summit experimental setup
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Power9, NVIDIA Tesla V100 GPUs

Operating systems and versions: Linux 4.14.0-115.14.1.el7a.ppc64le

Compilers and versions: GCC 8.1.1, CUDA 10.1.243

Libraries and versions: spectrum MPI library 10.3.1.2

Key algorithms: PREEMPT-HC, cuRipples

Input datasets and versions: Stanford Large Network Dataset Collection