A Population Data-Driven Workflow for COVID-19 Modeling and Learning

Jonathan Ozik,* Justin M. Wozniak,* Nicholson Collier,* Charles M. Macal,*
and Mickaël Binois†
* Argonne National Laboratory
† Inria Sophia Antipolis - Méditerranée

ABSTRACT

CityCOVID is a detailed agent-based model (ABM) that represents the behaviors and social interactions of 2.7 million residents of Chicago as they move between and colocate in 1.2 million distinct places, including households, schools, workplaces, and hospitals, as determined by individual hourly activity schedules and dynamic behaviors such as isolating because of symptom onset. Disease progression dynamics incorporated within each agent track transitions between possible COVID-19 disease states, based on heterogeneous agent attributes, exposure through colocation, and effects of protective behaviors of individuals on viral transmissibility. Throughout the COVID-19 epidemic, CityCOVID model outputs have been provided to city, county, and state stakeholders in response to evolving decision-making priorities, while incorporating emerging information on SARS-CoV-2 epidemiology. Here we demonstrate our efforts in integrating our high-performance epidemiological simulation model with large-scale machine learning to develop a generalizable, flexible, and performant analytical platform for planning and crisis response.

ACM Reference Format:
Jonathan Ozik,* Justin M. Wozniak,* Nicholson Collier,* Charles M. Macal,*, and Mickaël Binois†. 2021. A Population Data-Driven Workflow for COVID-19 Modeling and Learning . In *Proceedings of ACM Conference (Supercomputing '20).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 JUSTIFICATION FOR ACM GORDON BELL SPECIAL PRIZE FOR HPC-BASED COVID-19 RESEARCH

We

- implement a distributed COVID-19 ABM in Chicago, modeling 2.7 million individuals moving between 1.2 million places with hourly activity schedules;
- run mixed Bayesian calibration and ABM workflows at full machine scale and 84% utilization on ALCF Theta;
- provide forecasts and counterfactual analyses to city and state public health departments.

Supercomputing '20, November 16–19, 2020, Virtual

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 PERFORMANCE ATTRIBUTES

Attribute	Value
Category of achievement	Scalability & Time-to-solution
Type of method used	Bayesian optimization & Agent-based modeling
Results reported	Whole application including I/O
Precision reported	Discrete
System scale	Measured on full-scale system
Measurement mechanism	Utilization & Application metrics

3 OVERVIEW

The COVID-19 epidemic has brought epidemiological modeling to the forefront of discussions on how evidence-based decision making can be supported in times of crisis and uncertainty. Since the beginning of March 2020 we have applied CityCOVID, a detailed agent-based model (ABM) that represents the 2.7 million residents of Chicago in terms of people (behaviors and social interactions), places (1.2 million unique geolocations including households, schools, workplaces, and hospitals), and hourly activity schedules, with the aim of understanding COVID-19 transmission dynamics and potential effects of nonpharmaceutical interventions (NPIs). CityCOVID is based on the ChiSIM framework [50] and Repast HPC [20], which enable the creation of efficient, urban-scale MPI-distributed agent-based models (ABMs). Each individual, or agent, in the model includes its own individualized disease progression dynamic that determines transitions between possible COVID-19 disease states. Transitions have functional dependence on heterogeneous agent attributes, exposure through colocation over time with infected individuals, and other factors such as the effects of protective behaviors on viral transmissibility.

Our detailed modeling approach is substantially more computationally demanding than other types of modeling methods being applied to COVID-19, such as statistical models and compartmental models. However, in contrast to statistical models, which tend to be retrospective, we are able to investigate novel interventions that have not been implemented yet. When compared with compartmental models, we can simulate more specific and realistic NPIs, such as imposing restrictions on specific types of businesses, or detailed school attendance models, closely resembling those being considered by public health officials.

A central element in the application of epidemiological models is running in silico experiments for model calibration, scenario analyses, and uncertainty quantification, what we refer to as model exploration (ME) [61]. These processes often require sophisticated

machine learning (ML) algorithms to automate iterative, or sequential, strategic explorations of possible model behaviors, since brute-force approaches are not generally feasible even with the largest computing resources. Substantial advancements are being made in sequential ML algorithms that can be applied to ME. However, implementing dynamic ME processes at the necessary computational scales poses serious challenges, not the least of which is the need for efficiently dispatching and coordinating mixed workloads related to ML algorithms and complex models such as CityCOVID.

We developed the EMEWS framework [59], built on Swift/T [76], expressly to enable large-scale ME as ML-driven HPC workflows, which can be run on leadership-scale resources. Throughout the epidemic, members of our group have been regularly meeting with city, county, and state stakeholders and providing them with CityCOVID model outputs in response to their evolving priorities. Given the dynamic nature of information on the epidemiology of SARS-CoV-2 and the shifting foci of NPIs, we have engaged in a concerted effort to address computational efficiency across multiple dimensions and to reduce the "time to analysis."

The work we present here covers the approaches we took, including the following:

(1) Parallelization, load balancing, and caching within CityCOVID
(2) Applying multiple large-scale ML algorithms, such as multi-objective approximate Bayesian computation and Bayesian optimization, for model calibration and uncertainty quantification
(3) Coordinating single-node ML tasks and ordered multinode MPI tasks in a scalable HPC workflow
(4) Connecting the system to the real world via observed data and database access

We demonstrate the scaling performance of our integrated approach through whole-machine runs on the Argonne Leadership Computing Facility (ALCF) Theta supercomputer. Furthermore, we discuss the potential impact of this work in developing novel use cases for HPC resources beyond COVID-19 to a generally applicable decision-support platform.

4 STATE OF THE ART

4.1 Epidemiological Modeling

Current epidemiological modeling for disease spread is based on three intrinsically different types of modeling approaches: (1) so-called SIR/SEIR compartmental models, which are based on differential equations that specify mathematically rates of change between compartments, for example, susceptible (S), exposed but not infected (E), infected (I), and recovered (R) states; (2) agent-based models, in which the interactions of individual agents (people) and their explicit behaviors and contacts are directly simulated; and (3) statistical forecasting, which makes no assumptions about the details of the disease, such as the Institute for Health Metrics and Evaluation (IHME) model. Each of these approaches is capable of, for example, predicting the expected daily number of deaths, hospitalizations, and infections for COVID-19 in a given geographical region, as well as providing uncertainty bounds for these estimates, if stochastic elements are included in the model formulations.

The basic idea of "mechanistic" epidemiological models is to explicitly model the mechanisms of disease spread between individuals by modeling two processes: (1) the frequency of co-located contacts between individuals and (2) the nature of contacts between individuals that, probabilistically, facilitate or mitigate disease transmission. Agent-based models allow for these processes and all relevant variables to be included at the finest grain level of detail required to answer the questions posed by decision-makers. The CityCOVID model can simulate entire cities with millions of individuals, necessitating the use of HPC resources. The detailed agent-based modeling approach facilitates testing hypotheses concerning transmission, understanding optimal policies for reducing rates of transmission, and providing estimates for the impact of loss of staffing for critical services.

4.2 Agent-Based Modeling

ABMs and the toolkits with which they are developed, such as Repast Simphony [57], MASON [48], and NetLogo [74], have been confined to a single CPU for most of their development, with occasional forays into multithreaded parallelism. Within the past decade, however, we have seen increasing support for multiprocess distributed toolkits. Here, performance is achieved primarily by dividing the global population of agents among processes. A typical simulation can loop through all the agents during each iteration of a simulation. The fewer agents to iterate over, the faster the simulation. Toolkits such as D-MASON [23] initially attempted to avoid the complexity of MPI (at least in the Java environment) in distributing and coordinating agents across processes in favor of sockets and the Java Message Service before turning to MPI as a more performant solution for interprocess communication. Other ABM toolkits such as Pandora [67], Care HPS [15], and Repast HPC [20] also use MPI as the message-passing layer, while Flame GPU [65] distributes agents across a GPU such that their behavior can be executed in parallel. Other areas of emphasis are strategies for load balancing and the optimal partitioning of agents using different topologies and ghosting techniques [15, 21].

4.3 Model Exploration

The rise of artificial intelligence has provided broad and renewed interest in the development of statistical and ML algorithms that can address the evolving scale of ME tasks, resulting in vibrant ecosystems of free and open source libraries that are continually added to and updated as research frontiers are expanded. The relevant algorithms applicable to ME include various active learning [69] approaches, including Bayesian optimization (BO) approaches using Gaussian process (GP) surrogate models [11, 12] and ML models such as random forest [16, 61]; approximate Bayesian computation (ABC) approaches [7, 37], including sequential Monte Carlo [8] and Incremental Mixture ABC [68]; evolutionary approaches, including single [39] and multiobjective [25] genetic algorithms; and data assimilation approaches, including ensemble Kalman filtering [31, 63] and particle filters [4, 34]. Libraries that provide cutting-edge implementations of these algorithms or the components from which to implement them include scikit-learn [62], scikit-opt [38], BOTorch [6], ABCpy [29], ELFI [47] and DEAP [32] in Python; caret [44], mlrMBO [14], randomForest [46], EasyABC [41], IMABC [51], DiceKriging [66], hetGP [10], laGP [35], and GPareto [13] in R. However, a vast majority of these libraries and algorithms have not

been developed specifically with HPC scales or batch sampling in mind. That is, batch sizes often stay lower than a dozen (e.g., [18]), especially for multiobjective Bayesian optimization [24]. Exceptions include [73], which uses an *ad hoc* filtering to select batches of a few hundred candidates *a posteriori*, without considering repeating experiments and thus decreasing the noise at evaluated configurations. In addition, approaches combining ABC and GP methodologies have also been considered (see, e.g., [53, 75]), although the approach we take here differs in that ABC is used to provide initial evaluations and the GP optimization process provides refining. Moreover, toolkits specifically aimed at model exploration on HPC resources exist (see, e.g., [3, 36]); however, they are not geared toward integrating external, multilanguage libraries or large-scale mixed ML and simulation workflows.

4.4 Workflow Integration and MPI Task Management

Coordination among the ML and ABM modules in this workflow requires deep integration of components written in Python, R, and C++/MPI. The overall workflow shares hardware resources for these systems and rapidly switches among them. The workflow generates parameters in R-based optimizers, distributes these parameters to C++/MPI simulations, and registers progress via database operations called via Python—all on the same resources. Thus, building this complex application requires a workflow system with the capability to manage in-memory user libraries and third-party packages such as scripting language interpreters.

Other workflow solutions are generally constructed to orchestrate Unix-like program invocations [79] or whole-job scheduler submissions [26, 71]; some recent systems support in-memory Python snippets [5], while others have been extended to handle more complex ensembles and heterogeneous compute resources [27]. To our knowledge Swift/T is the only system that offers innate support for calling functions in Python and R, as well as tasks that are libraries that use MPI. In addition to coordinating ML and simulation tasks, our approach further allows for external ME algorithms to control the overall progression of complex workflows (see § 5.3).

Launching large numbers of MPI jobs is difficult on today's HPC systems. Typical solutions treat this as a shell programming problem and use a shell-scripting-like technique (or Python, for example) to manage calls to vendor-supplied job launchers (mpiexec, srun, aprun, jsrun, etc.), each with different command line interfaces. In contrast, the Swift/T approach seeks to use a pure MPI model for managing workflow tasks that use MPI. More generally, an MPI-based solution is needed [77].

As a quantitative comparison of these approaches, consider running a minimal 64-process C/MPI application that simply does `Init`, `Barrier`, and `Finalize`. On a Theta compute node, 10 executions of this program with aprun take 49.88 seconds for a rate of 0.20 tasks/second. In the Swift/T model with in-memory MPI tasks, however, 10 iterations take 0.33 seconds for a rate of 30.49 tasks/second—150× faster. This is due to the fact that the very notion of starting a child MPI task is very different in these two approaches. In the aprun model, many new processes must be created and connected, with the help of underlying job start services. In the Swift/T model, the child MPI task is simply called as a library on resources allocated within an pre-existing MPI job, by allocating resources managed and indexed by Swift/T and connecting them with a new subcommunicator.

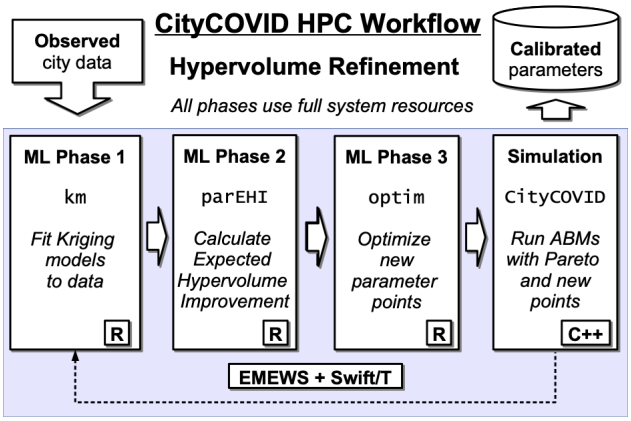

Figure 1: Conceptual progress model in the HVR workflow. The workflow loops until convergence.

5 INNOVATIONS REALIZED

The innovations within this work are in two areas:

(1) distributed simulation with CityCOVID and
(2) application and workflow integration of large-scale machine learning.

A schematic for the CityCOVID Hypervolumne Refinement (HVR) workflow, our most complex workflow for which we provide full machine performance results in § 7, is shown in Figure 1. Empirical data from hospital systems and public health departments are used to parameterize CityCOVID and to provide targets for calibration. Each iteration of the HVR algorithm (described in full in § 5.2.2) consists of three ML phases, each using an ensemble of single-node, many-core R methods distributed across the machine. The first phase is for fitting Gaussian process, or *kriging*, metamodels to the evaluated simulation runs, where evaluations are errors in fitting both hospitalization and death time series model outputs to empirical data. The second phase calculates the Expected Hypervolume Improvement of new candidate parameter points, and the third phase optimizes the best new candidate points. The calculations from the three ML phases produce simulation allocations, which are assigned across the existing Pareto front and new candidate points for evaluation. These simulations are instantiated and distributed across the system as multi-node C++/MPI programs. The HVR algorithm proceeds through multiple iterations as it refines the Pareto front (i.e., jointly optimizes multiple objectives). Simulation parameters are written to a Postgres database as the algorithm progresses for real-time feedback and final reporting.

5.1 CityCOVID

5.1.1 Model Overview. Models built with the ChiSIM framework [50] take three main inputs: **1)** synthetic people, **2)** places, and **3)** activity schedules. In the case of CityCOVID, we model

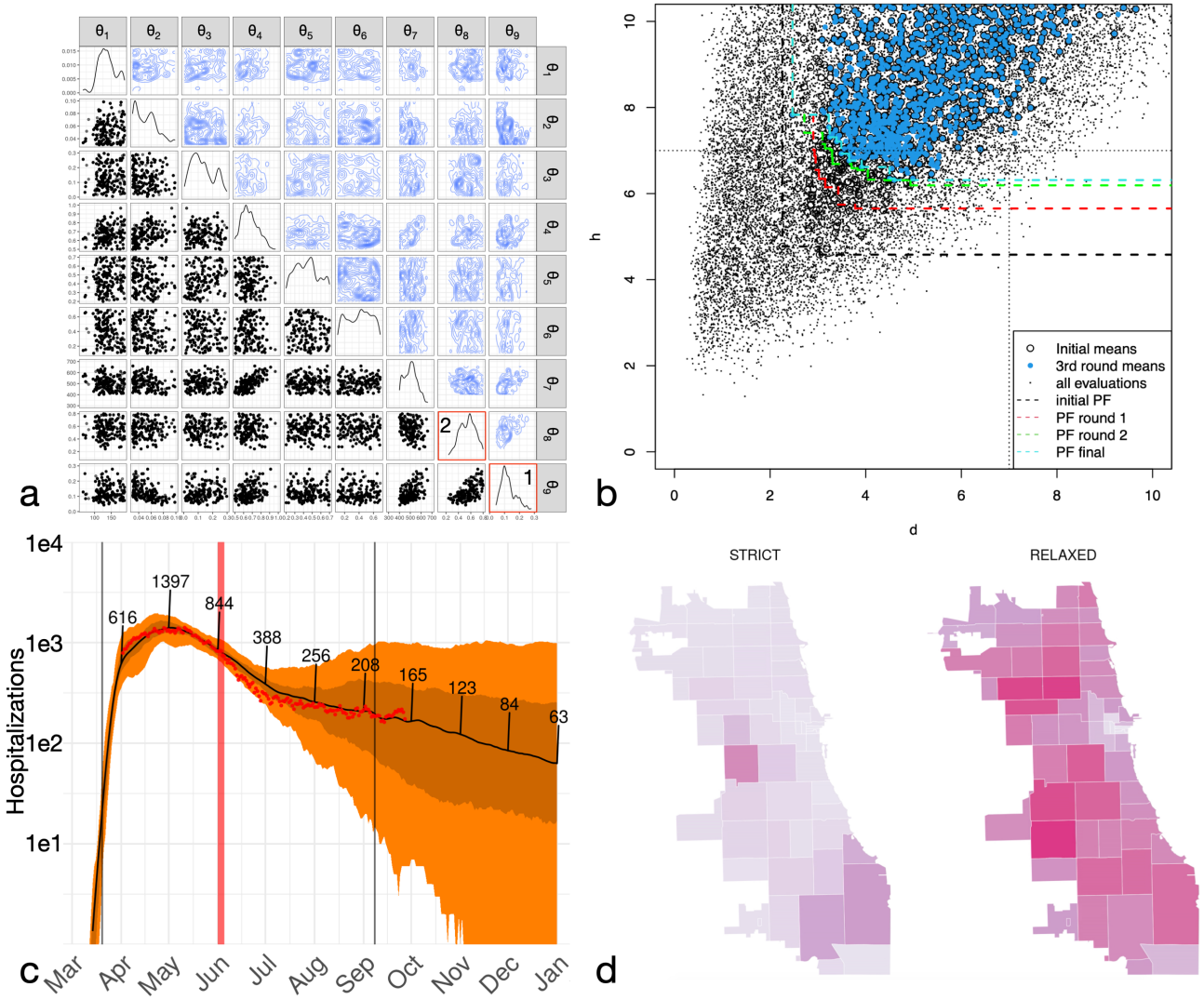

Figure 2: (a) Joint posterior distributions of CityCOVID input parameters (Table 1) from sequential ABC, (b) successive Pareto fronts of errors in deaths (x-axis) and hospitalizations (y-axis) from HVR workflow, (c) COVID-19 attributed hospitalization outputs from CityCOVID (red dots: empirical Chicago data, dark line: median simulation output, dark band: 50% simulation intervals, light band: 95% simulation intervals), (d) CityCOVID zip code level snapshot of weekly infection outputs at 47 days after June 3, 2020, initial easing of restrictions in Chicago for two scenarios (strict: population wide adherence to protective behaviors, i.e., θ_9 is maintained as reopening occurs, relaxed: gradual increase of θ_9 to a value corresponding to 80% viral transmission reduction).

the 2.7 million residents of Chicago as they move between 1.2 million places based on their hourly activity schedules. The synthetic population of agents extends an existing general-purpose synthetic population [17] and statistically matches Chicago's demographic composition. Agents can colocate at the geolocated places, which include households, schools, workplaces, hospitals, and group quarters (e.g., nursing homes, dormitories, jails). The agent hourly activity schedules are derived from the American Time Use Survey and the Panel Study of Income Dynamics and assigned based on agent demographic characteristics. CityCOVID includes COVID-19 disease progression within each agent, including differing symptom severities, hospitalizations, and age-dependent probabilities of transitions between disease stages.

In the present work we focus on the calibration of the CityCOVID parameters $\boldsymbol{\theta}$ listed in Table 1. These parameters were chosen through a sensitivity analysis described in § 5.2.1 and govern aspects ranging from the initial number of exposed individuals with which the model is seeded (θ_1) to the reduction in virus transmission due to individual protective behaviors such as mask wearing (θ_9). Model outputs are compared against two empirical data sources obtained through the City of Chicago data portal [19]: **1)** daily census

of hospital beds occupied by COVID-19 patients and **2)** COVID-19 attributed death data in and out of hospitals, both for residents of Chicago.

θ	$\pi(\theta)$	**Description**
θ_1	$U(60, 190)$	Initial number of exposed (infected but not infectious) agents at the beginning of the simulation
θ_2	$U(0.03, 0.1)$	Base hourly probability of transmission between one infectious and one susceptible person occupying the same location
θ_3	$U(0, 0.3)$	Magnitude of seasonality effect
θ_4	$U(0.5, 1)$	Per person probability of infection scaling factor due to ratio of infectious versus susceptible people in a location
θ_5	$U(0.2, 0.7)$	Effective infectivity during isolation in household
θ_6	$U(0.1, 0.7)$	Effective infectivity during isolation in nursing home
θ_7	$U(300, 700)$	Simulation time (hrs) corresponding to March 27, 2020
θ_8	$U(0.1, 0.8)$	Reduction in out of household (OOH) activities
θ_9	$U(0.01, 0.3)$	Reduction in transmission due to individual protective behaviors

Table 1: CityCOVID calibration parameters θ and priors $\pi(\theta)$.

5.1.2 Model Implementation. CityCOVID implements detailed agent behaviors and individualized disease progression and differentiates the movement of agents to places with how they behave when colocating with others in those places. For example, agents may engage in protective behaviors such as wearing a mask or keeping a six-foot distance, thereby reducing viral transmissibility while participating in activities outside of their homes. CityCOVID contains explicit representations of people and places, implemented as C++ classes. Places are assigned to a process rank, and persons move between places, and thus between processes, according to their hourly activity schedules. When a person is moved between processes, it needs enough of its state available on the target process to continue to follow its activity schedule and to transition between disease states.

The potential performance costs of this model are threefold: **1)** serializing of a person's state on the source process and deserializing of that state on the destination process; **2)** MPI-related overhead in transferring the person's state; and **3)** uneven distribution of persons among processes that causes uneven work loads. For the last factor, processes with less load (i.e., those with fewer persons to transition between disease states) would be idle waiting for those with greater loads to finish, resulting in longer runtimes. For **1)**, CityCOVID (via ChiSIM [21, 50]) mitigates the cost by caching person objects and their static attributes on every process that the person has visited. This minimizes the amount of data transferred and avoids any overhead associated with the frequent creation of new C++ objects and the concomitant memory allocation. For **2)** and **3)**, CityCOVID minimizes the amount of cross-process movement and load balances the number of persons on each process through graph partitioning, where each place is a vertex and each edge between two vertices represents the volume of person movement between those two places. The edge weight represents the number of persons that travel between the two places, and a vertex weight the total number of persons to visit that place. By partitioning this graph into groups of vertices (places), such that the edge weights connecting these groups are minimized and the vertex weights are roughly equal among all the partitions, highly connected places are placed on the same process, and we avoid any issues caused by uneven computational load. For the full Chicago scenario used here, the places graph consists of approximately 1.2 million vertices (places) and approximately 1.9 million edges (trips between places). To partition the graph, we used the METIS toolkit [43], assigning each place to the process rank corresponding to its graph partition. This assignment was performed during the development of the synthetic population (the collection of agents, places, and activity schedules), external from any simulation execution. While the agent behavior is itself stochastic, the number of places that an agent can select from as part of this behavior is tightly bounded, and thus effective load balancing using graph partitioning can be performed as a preprocessing step rather than during the simulation itself.

We also eliminated the time spent reading the input data (approximately 2.5 minutes on Theta) by caching it after the first read. The input data reflects the load balancing and assigns persons and places to particular ranks. Each rank caches the data assigned to it. Consequently, we ensure that when Swift/T allocates a group of 256 processes for each simulation run that the rank assignment within each group is repeatable.

5.2 Model Exploration Algorithms

5.2.1 Sequential ABC. The initial workflow we constructed was aimed at parameter estimation of so-called deep model parameters, those that are not readily observable. For example, the base hourly probability of transmission between one infectious and one susceptible person occupying the same location (θ_2 in Table 1) is difficult to measure directly. Similarly, the reduction in transmission from individual protective behaviors, such as wearing a mask and social distancing (θ_9 in Table 1), not only is difficult to directly measure but also can vary over time as behaviors change. These parameters, along with their joint distributions and uncertainties, can nonetheless be estimated through simulation-based inverse modeling approaches.

For simulators with intractable likelihoods, approximate Bayesian computation (ABC) methods, while computationally intensive, provide the ability to generate simulated posterior distributions of the model input parameters and, consequently, posterior model outputs that propagate data and model uncertainties to model results [7]. There are a number of ABC methods and libraries implementing them [29, 41, 47, 51]. Sequential ABC methods [8, 55, 68] improve the yield of parameter space sampling compared with one-shot rejection sampling approaches. In addition, batch sequential ABC methods can exploit the concurrency available on HPC resources, providing the ability to tune the batch sampling size to balance sample yield and time to solution.

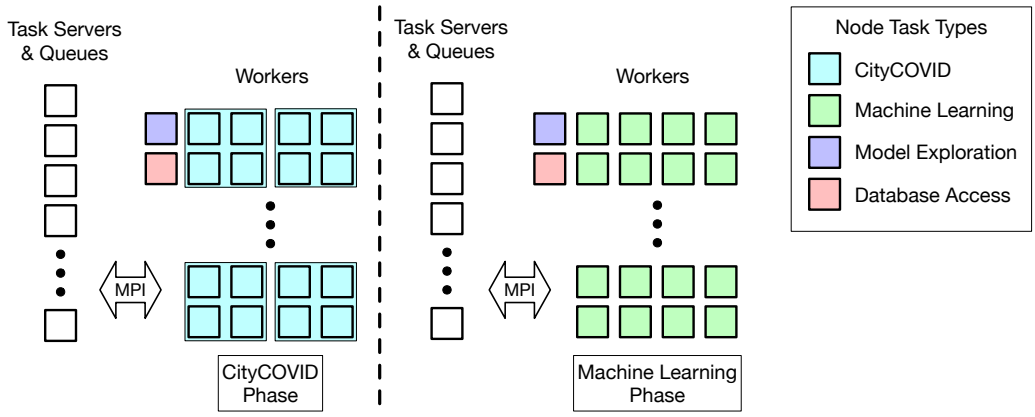

Figure 3: Application architecture for HVR workflow.

We utilized a batch sequential ABC algorithm [45] implemented in the EasyABC library [41], and we directly integrated it into an EMEWS workflow for our initial model calibration. The calibration was preceded by parameter prioritization analyses using an efficient global sensitivity method [56], identifying the nine parameters $\boldsymbol{\theta} = \theta_1, \ldots, \theta_9$ most relevant to the empirical data that we were attempting to match, time series of COVID-19 attributable hospitalizations $T_H(t)$ and deaths $T_D(t)$ for residents of Chicago, with corresponding simulation outputs $\tilde{T}_H(t, \boldsymbol{\theta}, \xi)$ and $\tilde{T}_D(t, \boldsymbol{\theta}, \xi)$, where ξ is a random variable encapsulating the random stream for each simulation replicate. Since one of the primary goals for using the model results was in creating forecasts, we used an exponentially weighted error function $L(\boldsymbol{\theta}, T_i, \tilde{T}_i, d), i \in \{H, D\}$, with daily discount rate d tuned to 98% and 95% for H and D, respectively, and averaged over replicates for each $\boldsymbol{\theta}$ (shortened to $L_i(\boldsymbol{\theta})$ hereafter). The ABC algorithm starts by sampling from the parameter priors $\pi(\boldsymbol{\theta})$ and iteratively creates improved approximations of the posterior distribution of the nine model parameters $p(\boldsymbol{\theta}|T_H, T_D)$ [45]. The final simulated posterior distribution based on the time period from the beginning of the epidemic to the initial lifting of the stay-at-home order in Chicago on June 3, 2020, is shown in Figure 2a.

The diagonal panels in Figure 2a are the marginal distributions for each input parameter θ_i, and the off-diagonal elements show contour (upper) and scatter plots (lower) of each two-dimensional parameter subspace $\theta_{i,j}$. The calibration resulted in two main takeaways, both associated with agent behaviors during the stay-at-home period: **1)** the reduction in transmission due to individual protective behaviors (θ_9) exhibited a pronounced peak at about 90% transmission reduction (Figure 2a inset 1), and **2)** the average proportion of agent pre-COVID-19 out-of-household activities where agents instead opted to stay at home during the shutdown (θ_8) showed a broader range of values between 40 and 60% (Figure 2a inset 2). While the second result can be compared with empirical data on mobility based on cellphone data, providing constraints on the parameter estimation result, the first result encapsulates generally unobserved dynamics of person-to-person interactions across the population in response to public health messaging and perceptions of risk.

As the pandemic progressed, our team sought methods for improving the time to solution for generating parameter estimates. One such approach was utilizing another sequential ABC algorithm, IMABC [68], which members of our team implemented as an R package [51]. IMABC provided two main improvements over the original ABC algorithm. Because of a more directed sampling approach, IMABC increased the efficiency of sampling from the CityCOVID parameter space, which sped up algorithm convergence. The IMABC algorithm and package also included the ability to continue sampling from a checkpointed algorithm state, that is, to further refine a previously generated simulated posterior distribution. This allowed for improved robustness, for example., for restarting after HPC resource failure, and the possibility for optionally longer-running calibrations when convergence in the standard timeframe proved difficult.

Another approach we used for improving parameter estimation time to solution was to employ metamodels, also known as surrogate models, for multiobjective optimization, which we discuss next.

5.2.2 Hypervolume Refinement. To complement the uncertainty estimation in the parameter space provided by the ABC approach, we also consider finding a set of best parameters matching simultaneously both COVID-19 attributable deaths and hospitalizations. Denote L_D (resp. L_H) : $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^9 \mapsto \mathbb{R}$ the corresponding calibration errors where $\Theta := \text{supp}(\pi(\boldsymbol{\theta}))$. Here, given the intrinsic variability of ABM outputs across runs for the same parameters, the multiobjective optimization (MOO) problem is to obtain

$$\min_{\boldsymbol{\theta} \in \Theta} (\mathbb{E}_\xi[L_D(\boldsymbol{\theta})], \mathbb{E}_\xi[L_H(\boldsymbol{\theta})]), \tag{1}$$

as commonly done for simulation experiments (see, e.g., [40]). In general, no solution minimizes all objectives at once; instead, the Pareto dominance relation is used to define optimality. A solution is said to be optimal (or nondominated) if no other solution is at least as good for all objectives and strictly better for one. The set of all optimal solutions in the objective space is called the Pareto front, and the aim of MOO is to find an accurate discrete representation of it. Directly solving the MOO problem is out of reach even on the largest HPC resource because of the large number of runs it would involve. Instead, the Bayesian optimization paradigm is used (see, e.g.,

[33, 70]), where probabilistic surrogate models of the objectives are used to sequentially select new evaluations based on an acquisition function. As surrogates, Gaussian processes are popular for their ability to provide uncertainty on their prediction (see, e.g., [64]). For MOO, an adapted acquisition function performing a trade-off between exploration of unknown regions and exploitation of promising ones is the Expected Hypervolume Improvement (EHI) [30]. It is based on the hypervolume metric, that is, the volume added to the current Pareto front estimation, given a reference point. The approach is implemented, for instance, in the R package GPareto [13], but requiring adaptations for the combination of low signal-to-noise ratio and massive batching enabled by the available concurrency involved here.

To handle the challenges associated with running hundreds of noisy simulations at once, we developed the multiobjective Bayesian optimization loop shown in Figure 1. To both accurately learn the input-dependent variance and reduce the computational complexity of GP modeling, we complement the exploitation side of BO by leveraging replication of runs at the same location. Every design $\boldsymbol{\theta}$ is replicated at least ten times to prevent an overly optimistic estimation of the Pareto front. Figure 2b depicts where individuals runs, shown as small black dots, can be much closer to the zero-error target than estimated means, shown as hollow circles and blue dots. Based on empirical means and variance of initial evaluations—provided here by the ABC computations—the first ML phase (ML Phase 1 in Figure 1) is to build one GP model per objective, with the DiceKriging package [66]. The optimization of the corresponding hyperparameters is performed in parallel. Note that instead of global GP models that can predict everywhere, local GP models can be constructed to predict at specific locations, for example, with [35]. The advantage of this second option is to be more decentralized. Then the expected hypervolume improvement is optimized with a multistart approach, since the EHI optimization landscape is expected to be highly multimodal. Specifically, in ML Phase 2, a large set of uniformly sampled designs is evaluated, from which the best candidates constitute starting points for local optimization in ML Phase 3. The reference point for hypervolume computations is fixed at (7,7), while the reference Pareto front to improve over is estimated based on the predicted means at the evaluated designs, hence filtering out the noise via the GPs. In order to further refine the current estimate of the Pareto front, part of the evaluation budget is dedicated to allocating more replication at the corresponding design. The main part of the simulation budget is for the best candidates found in ML Phase 3, where the number of replicates depends on the corresponding EHI value (larger potential of improvement gets more replicates). Models are updated as new simulation evaluations come in, until the maximum number of iterations. The result is provided in Figure 1b, where the initial estimation of the Pareto front (black dotted line) is overly optimistic. Then it keeps regressing before stabilizing and improving, suggesting that despite noise, the adaptive scheme evaluates more over the region of interest. By relying on standard but robust BO components amenable to parallelization, the HVR workflow thus propels multiobjective BO to whole-machine scales on ALCF Theta.

5.3 Machine learning in HPC

Running machine learning workloads in high-performance computing systems is at the core of the technological contributions of this effort. Our approach is to use Swift/T to coordinate and distribute in-memory library calls to ML and simulation codes in an HPC-oriented manner. While this technique had been prototyped earlier [58], the CityCOVID workload effort involved the development and integration of the following innovations.

5.3.1 Pluggable algorithm workflow control. A key innovation of EMEWS is the ability to directly incorporate R or Python-based model exploration algorithms. This is done by defining resident, or stateful, tasks to encapsulate the logic within iterating, state-preserving algorithms [60]. These in-memory calls are critical for ML-based workflows in which numerically oriented modules maintain state over long periods but are treated as discrete tasks by the workflow system. While certain workflow systems are designed from the ground up to perform numerical procedures such as optimization [1, 2], these are generally difficult to extend and reprogram for arbitrary workflows. We have exploited this capability to implement a variety of workflows specifically geared toward the concurrency afforded by HPC resources, both in terms of the number of simultaneous *simulation* tasks and through the mixed use of worker pools to handle *machine learning* tasks as well. We used sequential ABC approaches [45, 68] and HVR [9] for uncertainty quantification (Figure 2a) and calibration (Figure 2b). The ML workloads within the HVR workflow (Figure 1) were implemented by using R-based parallelization via a new `doEMEWS` adapter for the `foreach` package [54], which provides transparent batch dispatching of R code to the worker node pool (Figure 3). The calibrated model can generate a variety of epidemiologic outputs, including hospitalizations (Figure 2c), and zipcode-level infection rates (Figure 2d).

5.3.2 Load balancing and work distribution. From a computer systems perspective, a challenge for ML-based workflows is managing the differing resource requests from simulation and learning-based work items. In the CityCOVID/HVR workflow, simulation tasks require 256 ranks, and learning tasks require 64. The Swift/T model allows workflows to request tasks on any number of ranks from 1 to the number of workers allocated to the compute job. These tasks may optionally request that the ranks for a task occupy contiguous ranks starting at regular intervals or other constraints. For example, all CityCOVID workflows run the simulations on contiguous ranks starting on ranks such that rank mod $256 \equiv 0$. The ML tasks run independent R interpreters on each node of the system, each of which uses threads internally to utilize the 64 cores on each node. The database access tasks call into Python libraries using a single core at a time. Thus, Swift/T faces a significant work distribution challenge.

Swift/T translates workflows into ADLB [49] programs. In the implementation used by Swift/T, ADLB divides the available ranks into workers and servers. Any number of servers and workers may be used, down to a singleton of each. The workers simply perform work requested from their server. Servers exchange work using the work stealing approach from Scioto [28], in which empty servers steal half of the work queue from another server selected at random. This explains the exponential ramp-up commonly seen in Swift/T

performance plots. For the CityCOVID workload, we used increasing numbers of servers for increasing run sizes. Servers spent the bulk of their time looking for contiguous blocks of ranks for MPI tasks, and this work was easily shared by the work stealing approach.

5.4 Supporting Public Health

We have been using the CityCOVID model to forecast the spread of COVID-19 in Chicago and have been providing model results to support city and state public health officials and decision-makers. From the beginning, the decision-makers we work with have had the philosophy that their "decisions would be guided by data, science, and public health experts." To meet these requirements, decision-makers recruited data analytics support for understanding the current COVID-19 situation and epidemiological modeling support for forecasting what might happen in the future based on possible NPIs, and they paired these groups with public health departments.

We have supported the Chicago Department of Public Health (CDPH) since the early stages of the pandemic, March 2020 [72]. As described in § 5.2.1, we have been able to estimate the extent to which individual protective behaviors, such as mask wearing and social distancing, have affected community SARS-CoV2 transmission. These model-based insights were used by CDPH in their briefings with Chicago aldermen about the importance of maintaining COVID-19 mitigation measures in the lead up to the partial reopening on June 3, 2020.

We have also supported the Illinois Department of Public Health through the Illinois Governor's COVID-19 Modeling Task Force (IGCMTF) [52]. The IGCMTF includes four modeling groups, including Argonne, that are applying various epidemiological modeling approaches to understand COVID-19 spread and explore what can be done to reduce the spread.

For interacting with public health officials, we established a weekly/biweekly routine that consists of **1)** calibrating the model based on the most recent data; **2)** designing and running experiments with the model to forecast effects on the spread by varying scenario parameters; **3)** compiling and analyzing the model outputs from thousands of simulations; **4)** applying quality control/assurance checks, which include independent subject matters experts such as epidemiologists, to the assumptions and results and identifying possible anomalies; **5)** applying causal analysis to explain why the results came out as they did; and **6)** reporting the results in a structured format that public health officials can easily discern and understand the implications. A standardized set of charts and graphs is provided in as succinct a form as possible and presented in live forums to public health officials by the modelers.

Since the fall of 2020 and continuing through today, questions have focused on the effects of vaccination programs and, more recently, the effects of variants of concern, in other words, mutations that, for example, result in increased disease severity, transmissibility, and vaccine escape. The fine-grained nature of the CityCOVID model allows us to track which variant an individual agent has been infected with and follow the disease course and subsequent transmission of that individual, where parameters unique to that variant can be adjusted. Our group continues to calibrate and run the model and report results to public health officials regularly.

6 HOW PERFORMANCE WAS MEASURED

6.1 What applications were used to measure performance

All software was built with GCC 7.3.0 with the Cray MPI wrappers for Theta as appropriate. The CityCOVID model is based on the RepastHPC 2.3.1 agent-based modeling system and the ChiSIM Chicago model 0.4.2. The workflow system was Swift/T 1.4.3, which was linked to interpreters in Python 3.8.2.1, R 3.6.0, Tcl 8.6.6, and Cray MPI. All Python and R libraries used were the latest compatible with these interpreters as of October 1, 2020. The R library size to perform these optimizations contains 112 packages totaling 321 MB. In order to improve the performance associated with numerical operations in the machine learning tasks, an external R with Cray LibSci v.20.03.1 providing vendor-optimized BLAS/LAPACK libraries was also used.

6.2 System and environment where performance was measured

All experiments were run on the Theta system at the ALCF. Theta is a Cray XC40 with 4,392 compute nodes, each with an Intel KNL 7230 (Xeon Phi), aggregating 11.7 petaflops in total. Each node has 64 compute cores with access to 16 GB of high-bandwidth in-package memory, 192 GB of DDR4 RAM, and 128 GB of SSD. The system interconnect is a dragonfly network. The system has Lustre and GPFS filesystems; we used Lustre for software installations and GPFS for application data. We used the vendor-provided Cray MPICH 7.7.17.

6.3 How performance was measured

The technical details of our workflow solution are as follows. The workflow runs as a single job allocation in the Cobalt scheduler on Theta. The Swift/T runtime runs as a single MPI program across all nodes and evaluates the logic of the Swift workflow. Tasks in the workflow such as the invocation of R code snippets (the ML algorithms) or MPI-enabled libraries (CityCOVID/C++) are placed in a distributed task queue in the ADLB library underlying Swift/T. Even database accesses are represented as Python code snippets (via Psycopg2); thus, they do not block the progress of the overall workflow. MPI tasks are launched by Swift/T using MPI 3.0 features developed previously [78].

Performance was simply measured by using the clock libraries available in the programming models used and reported in log files. The precision of our measurements need be accurate only to the nearest second to describe the performance. The overall performance metric of our workflow is delivering high utilization to the compute-intensive ML and ABM modules.

7 PERFORMANCE RESULTS

In this section, we present five types of performance results: an illustration of a full-scale run on Theta (§ 7.1), strong-scaling results for full-system scale (§ 7.2), time to solution results for the learning tasks (§ 7.3), communication and messaging rates (§ 7.4), and ABM-specific performance results (§ 7.5).

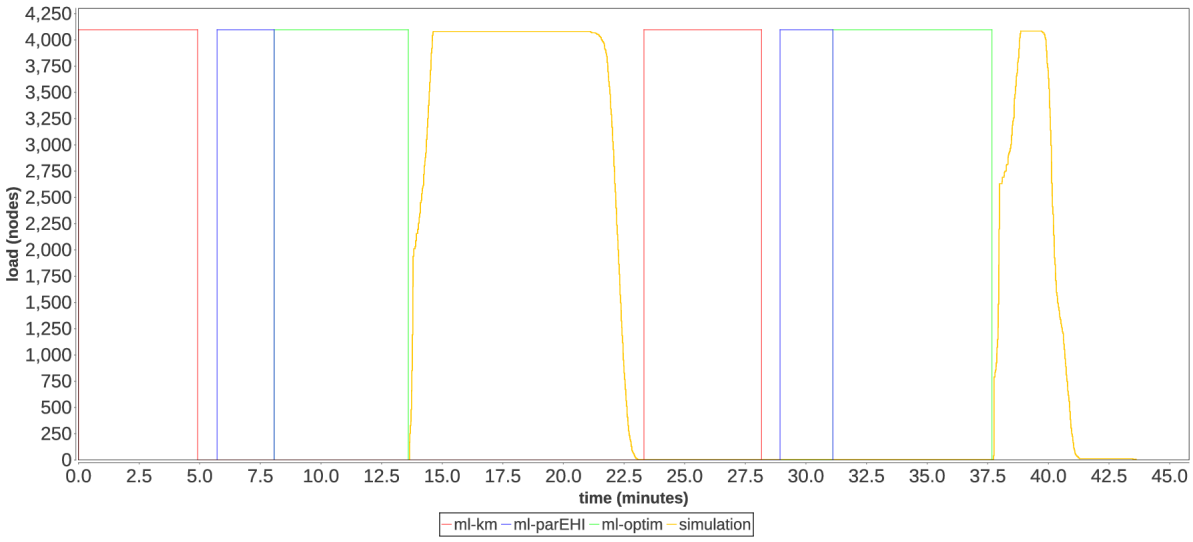

Figure 4: Utilization timeline on 4,098 nodes of Theta consisting of ML and simulation phases.

7.1 System utilization at full scale

Figure 4 shows a full-scale run of the HVR workflow on 4,098 nodes of Theta. Each colored phase corresponds to a phase shown in Figure 1. Two optimization iterations are shown from the beginning of a run.

A key performance challenge of this workflow is juggling the 2,040 MPI simulation tasks generated by the workflow, each a 256-core execution. As parameters for these tasks emerge from the HVR algorithm, this work must be rapidly distributed. Additionally, there are constraints on the task placement, since the MPI task should occupy contiguous cores across whole nodes to support fast communication and a data caching strategy used by CityCOVID. Nevertheless, the worker utilization across the two optimization iterations, including the learning and simulation tasks, is 84%. This is a good result considering that the Swift/T workflow scheduler operates at the rank granularity, and that 128 server ranks were distributing short (sub-minute), multi-process tasks to 262,144 worker ranks.

The plot shows fine-grained load levels for the MPI-based simulation phase. As shown, the workflow rapidly ramps up as the simulation phase begins and ramps down more slowly as tasks exit. The first phase takes 9.4 minutes, whereas the second phase takes 5.9 minutes, demonstrating the benefit of the caching strategy. The ML task utilization is not measured directly; rather, we measure the overall time to solution in the following.

7.2 Strong scaling

The next key metric of our workflow is demonstrating strong scaling for a given problem as the computing allocation size increases. We ran the same workflow parameters on Theta in job node counts 257, 513, 1,025, 2,049, and 4,098. We then measured the overall time to solution and calculated the corresponding task rates. This corresponds to 2 nodes or 128 ranks of ADLB servers (§ 5.3.2) for the largest case, and 1 node or 64 ranks of ADLB servers for the other cases. The remaining nodes were allocated as ADLB workers.

As shown in Figure 5, the time to solution and simulation rate both accelerate to full scale on Theta, and the likelihood of further scaling is encouraging. In our smallest case the workflow takes 5 hours and 54 minutes on 257 nodes, but at 4,098 nodes it completes in under 44 minutes!

Further utilization improvements could be possible with asynchronous HVR algorithms, enabling overlap between workflow phases. The Swift/T programming model supports arbitrary data dependencies, enabling such patterns; and the flexibility of being able to directly integrating Python and R code will facilitate such algorithmic investigations.

7.3 ML phase scaling

Here we report the scaling results for the ML phases. The ML phase uses the novel `doEMEWS` module to distribute R-based workloads via Swift/T single-node task distribution. For each run size up to full system scale, the number of tasks produced in each ML phase is the same. Thus, the most important metric is time to solution per phase.

In Figure 6 we report the average time per phase for the ML workload. The most time-consuming phase is the kriging model (ml-km) phase, which takes 177 seconds on 256 worker nodes and only 19 seconds on 4,096 worker nodes. The least time-consuming phase is the Pareto front improvement phase (ml-parEHI), which takes 147 seconds on 256 worker nodes and 19 seconds on 4,096 nodes. As shown in the plot, scaling efficiency drops off at the larger scales because the system runs low on tasks to execute, allowing the task distribution and task runtime imbalance overheads to negatively impact utilization.

7.4 Messaging rates

Workflow variables in Swift/T are stored on the task servers and accessed as in a distributed hash table over MPI. Over the whole workflow, in the 4,098-node run the system performed 722,485 retrieves in 2,619 seconds, an aggregate rate of 276 data reads per

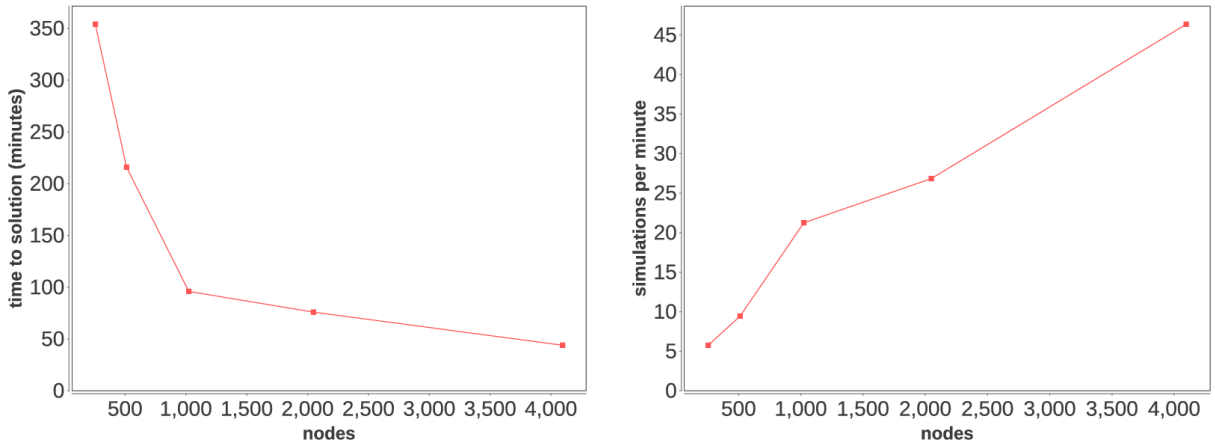

Figure 5: Overall time to solution and strong-scaling plot for HVR workflow.

second. During the peak period of the first simulation ramp-up, shown in Figure 4 near minute 13, the system performed 148,854 read accesses in 12.5 seconds, for a rate of 11,908 reads/second. This workload was spread over 128 task servers, for a rate of 93.0 accesses per second.

7.5 ABM-specific performance results

Within the HVR workflow, each CityCOVID model ran for 69 simulated days (1,656 hours), corresponding to approximately 5×10^9 agent-hours. The significance of this metric is that it tracks how many hours agents had the potential to colocate with other agents and spread or be exposed to COVID-19. At the 4,098-node scale and after the input data caching had occurred, 1,020 concurrent simulations were launched and completed in 5.9 minutes. This corresponds to the simulation of 1.3×10^{10} potential transmissions/second. The runtime of the CityCOVID model could potentially be improved by parallelizing disease transmission and disease state transitions, either by exploiting intraprocess parallelism (threading) or reimplementing disease states and disease state transitions to be more GPU amenable. Both of these potential improvements are complicated by the necessity of logging disease states and transitions and thus could require a parallel logging capability. We have begun exploring how to leverage GPUs in an agent-based modeling toolkit with our work on Repast4Py [22], our next-generation distributed ABM toolkit in Python.

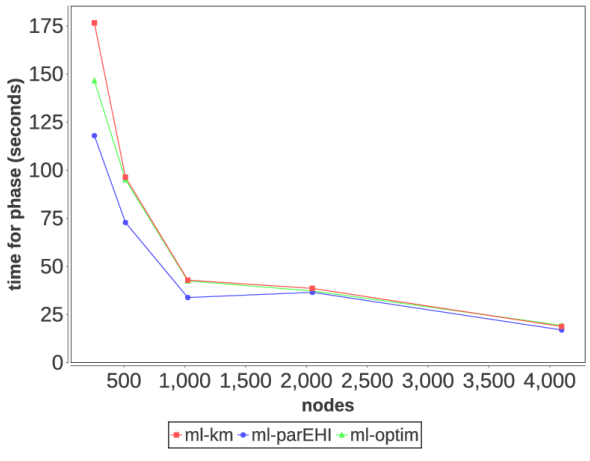

Figure 6: Time to solution for ML phases by node count.

8 IMPLICATIONS

The implications of the success of this workflow applications are broad. Here we describe the implications for COVID-19 modeling and public health, as well as for data science, learning, and simulation in the HPC space.

8.1 COVID-19 modeling and public health

Public health departments need guidance for prioritizing interventions in the context of limited resources and empirical data uncertainties. Through our regular meetings with city, county, and state stakeholders during the COVID-19 pandemic, we have first-hand experience in the dynamic nature of evolving mitigation priorities and the need for the detailed modeling that CityCOVID provides to rapidly coevolve. CityCOVID can simulate specific and realistic NPIs closely resembling those being considered by public health officials and can thereby guide local policy and intervention development. By integrating high-performance epidemiological simulation with large-scale machine learning, we have developed a generalizable, flexible, and performant analytical platform that can meaningfully support evidence-based decision-making *during* a public health crisis.

Furthermore, the COVID-19 pandemic has highlighted a number of issues, including drastic health inequities. Reducing COVID-19 morbidity and mortality will likely require an increased focus on the social determinants of health, given their disproportionate impact on populations most heavily affected by COVID-19. The detailed data-driven modeling approach that CityCOVID represents, coupled with the HPC solution developed here, has the potential to provide

the robust exploration of underlying factors and reveal promising targeted interventions to reduce these observed inequities. The melding of advanced simulation, machine learning, and high-performance computing enables an *in silico* laboratory for identifying gaps where additional empirical data are needed and facilitates hypothesis generation.

8.2 Convergence of data science, learning, and simulation for HPC

This application demonstrates the capability and need to bring HPC closer to diverse scientific communities, including the public health sector. By using detailed local data as a starting point to parameterize and calibrate simulations and produce actionable analyses (i.e., from observational data to scenario outputs documented in an integrated Postgres database), we applied Theta as the numerical engine at the center of a data pipeline. This made the overall workflow more relevant to our public health collaborators and demonstrated the capability of the system to be an integral part of the decision-making process.

From a computer systems perspective, the complex integrations and high performance achieved here demonstrate that applications written in high-level workflow languages and mathematical systems can be executed at large scale. Additionally, they can be closely integrated with previously developed MPI-based scientific applications that are scalable in themselves. We also demonstrated that very large (thousands) ensembles of MPI program executions can be run on the system without negatively impacting the scheduler or other users by tapping into the capabilities of MPI, via the high-level model offered by Swift/T.

We expect that data+learning+simulation workflows will become more tightly integrated in the near future. As shown in Figure 4, learning is already half of the job. The capabilities shown here are initial steps to more general-purpose HPC-enabled decision-support platforms that can rapidly assess and forecast the course of disease outbreaks and other crises. We expect that incorporating real-time data streams and weather-forecast-like automated steering will drive further enhancements and increase the performance and robustness of the results obtained.

In the broad ME context of Gaussian process-based Bayesian optimization, parallel infill criteria have been developed to benefit from parallelism of modern architectures. Yet, most of these works cannot be directly transposed to HPC scale. More work is needed to leverage existing methods to scale GPs with millions of fixed observations to the sequential framework. Thus, developing algorithms that are specifically geared toward and can take advantage of HPC resources has the potential to advance computational science and open up previously unexplored areas of statistical research. There are trade-offs to consider, such as between accurately selecting the optimal points for a given task and the time it takes to make this selection at the specified accuracy. Besides the ability to tackle more complex problems with an increasing number of variables, higher concurrency raises challenges in coping with consideration of synchronous versus asynchronous algorithms. Support for asynchronous algorithms (see, e.g., [42]) is a promising perspective for future work.

ACKNOWLEDGMENTS

This research was supported by the DOE Office of Science through the National Virtual Biotechnology Laboratory, a consortium of DOE national laboratories focused on response to COVID-19, with funding provided by the Coronavirus CARES Act. This work was also funded by an award from the c3.ai Digital Transformation Institute and the National Institute of Allergy and Infectious Diseases grant R01AI136056.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. We would like to thank the Argonne Leadership Computing Facility staff for their timely and critical support. This research was completed with resources provided by the Laboratory Computing Resource Center at Argonne National Laboratory.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

REFERENCES

[1] Abramson D, Lewis A, Peachey T and Fletcher C (2001) An automatic design optimization tool and its application to computational fluid dynamics. In: *Proc. SuperComputing*.

[2] Adams B, Bauman L, Bohnhoff W, Dalbey K, Ebeida M, Eddy J, Eldred M, Hough P, Hu K, Jakeman J, Stephens J, Swiler L, Vigil D and Wildey T (2014) Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.0 user's manual. Sandia Technical Report SAND2014-4633, Updated November 2015 (Version 6.3).

[3] Adams BM, Ebeida MS, Eldred MS, Jakeman JD, Swiler LP, Stephens JA, Vigil DM, Wildey TM, Bohnhoff WJ, Eddy JP and others (2014) Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[4] Arulampalam M, Maskell S, Gordon N and Clapp T (2002) A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing* 50(2): 174–188. DOI:10.1109/78.978374.

[5] Babuji Y, Brizius A, Chard K, Foster I, Katz DS, Wilde M and Wozniak JM (2017) Introducing Parsl: A Python parallel scripting library. In: *Zenodo*. URL http://doi.org/10.5281/zenodo.853491.

[6] Balandat M, Karrer B, Jiang DR, Daulton S, Letham B, Wilson AG and Bakshy E (2020) BoTorch: A framework for efficient monte-carlo bayesian optimization. In: *Advances in Neural Information Processing Systems 33*. URL https://proceedings.neurips.cc/paper/2020/hash/f5b1b89d98b7286673128a5fb112cb9a-Abstract.html.

[7] Beaumont MA (2019) Approximate Bayesian computation. *Annual Review of Statistics and Its Application* 6(1): 379–403. DOI:10.1146/annurev-statistics-030718-105212. URL https://doi.org/10.1146/annurev-statistics-030718-105212.

[8] Beaumont MA, Cornuet JM, Marin JM and Robert CP (2009) Adaptive approximate Bayesian computation. *Biometrika* 96(4): 983–990. DOI:10.1093/biomet/asp052. URL http://biomet.oxfordjournals.org.proxy.uchicago.edu/content/96/4/983.

[9] Binois M, Ginsbourger D and Roustant O (2015) Quantifying uncertainty on Pareto fronts with Gaussian process conditional simulations. *European Journal of Operational Research* 243(2): 386–394. DOI:10.1016/j.ejor.2014.07.032. URL http://www.sciencedirect.com/science/article/pii/S0377221714005980.

[10] Binois M and Gramacy RB (2019) hetGP: Heteroskedastic Gaussian Process Modeling and Design under Replication. URL https://CRAN.R-project.org/package=hetGP.

[11] Binois M, Gramacy RB and Ludkovski M (2018) Practical heteroscedastic Gaussian process modeling for large simulation experiments. *Journal of Computational and Graphical Statistics* 27(4): 808–821. DOI:10.1080/10618600.2018.1458625. URL https://doi.org/10.1080/10618600.2018.1458625.

[12] Binois M, Huang J, Gramacy RB and Ludkovski M (2019) Replication or exploration sequential design for stochastic simulation experiments. *Technometrics* 61(1): 7–23. DOI:10.1080/00401706.2018.1469433. URL https://doi.org/10.1080/00401706.2018.1469433.

[13] Binois M and Picheny V (2019) GPareto: An R package for Gaussian-process-based multi-objective optimization and analysis. *Journal of Statistical Software* 89(8): 1–30. DOI:10.18637/jss.v089.i08.

[14] Bischl B, Richter J, Bossek J, Horn D, Thomas J and Lang M (2017) mlrMBO: A modular framework for model-based optimization of expensive black-box functions. *arXiv:1703.03373 [stat]* URL http://arxiv.org/abs/1703.03373. ArXiv: 1703.03373.

[15] Borges F, Gutierrez-Milla A, Luque E and Suppi R (2017) Care HPS: A high performance simulation tool for parallel and distributed agent-based modeling. *Future Generation Computer Systems* 68: 59–73. DOI:https://doi.org/10.1016/j.future.2016.08.015. URL http://www.sciencedirect.com/science/article/pii/S0167739X16302758.

[16] Breiman L (2001) Random forests. *Machine Learning* 45(1): 5–32. DOI:10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[17] Cajka JC, Cooley PC and Wheaton WD (2010) Attribute assignment to a synthetic population in support of agent-based disease modeling. *Methods report (RTI Press)* 19(1009): 1–14. DOI:10.3768/rtipress.2010.mr.0019.1009. URL https://pubmed.ncbi.nlm.nih.gov/22577617.

[18] Chevalier C and Ginsbourger D (2013) Fast computation of the multi-points expected improvement with applications in batch selection. In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 59–69.

[19] City of Chicago (2021) Data Portal. URL https://data.cityofchicago.org/.

[20] Collier N and North M (2013) Parallel agent-based simulation with Repast for high performance computing. *SIMULATION* 89(10): 1215–1235. DOI:10.1177/0037549712462620. URL https://doi.org/10.1177/0037549712462620.

[21] Collier NT, Ozik J and Macal CM (2015) Large-scale agent-based modeling with repast HPC: A case study in parallelizing an agent-based model. In: *Euro-Par 2015: Parallel Processing Workshops - Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers*. pp. 454–465. DOI:10.1007/978-3-319-27308-2_37. URL http://dx.doi.org/10.1007/978-3-319-27308-2_37.

[22] Collier NT, Ozik J and Tatara ER (2020) Experiences in developing a distributed agent-based modeling toolkit with Python. In: *2020 IEEE/ACM 9th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*. pp. 1–12. DOI:10.1109/PyHPC51966.2020.00006.

[23] Cordasco G, Mancuso A, Milone F and Spagnuolo C (2014) Communication strategies in distributed agent-based simulations: The experience with D-Mason. In: an Mey D, Alexander M, Bientinesi P, Cannataro M, Clauss C, Costan A, Kecskemeti G, Morin C, Ricci L, Sahuquillo J, Schulz M, Scarano V, Scott SL and Weidendorfer J (eds.) *Euro-Par 2013: Parallel Processing Workshops*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-54420-0, pp. 533–543.

[24] Daulton S, Balandat M and Bakshy E (2020) Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. *Advances in Neural Information Processing Systems* 33.

[25] Deb K, Pratap A, Agarwal S and Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6(2): 182–197. DOI:10.1109/4235.996017.

[26] Deelman E, Singh G, Su MH, Blythe J, Gila Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, Laity A, Jacob JC and Katz DS (2005) Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 13.

[27] Di Natale F, Bhatia H, Carpenter TS, Neale C, Kokkila-Schumacher S, Oppelstrup T, Stanton L, Zhang X, Sundram S, Scogland TRW, Dharuman G, Surh MP, Yang Y, Misale C, Schneidenbach L, Costa C, Kim C, D'Amora B, Gnanakaran S, Nissley DV, Streitz F, Lightstone FC, Bremer PT, Glosli JN and Ingólfsson HI (2019) A massively parallel infrastructure for adaptive multiscale simulations: modeling RAS initiation pathway for cancer. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-4503-6229-0, pp. 1–16. DOI:10.1145/3295500.3356197. URL https://doi.org/10.1145/3295500.3356197.

[28] Dinan J, Krishnamoorthy S, Larkins DB, Nieplocha J and Sadayappan P (2008) Scioto: A framework for global-view task parallelism. *Intl. Conf. on Parallel Processing* 0: 586–593. DOI:10.1109/ICPP.2008.44.

[29] Dutta R, Schoengens M, Onnela JP and Mira A (2017) ABCpy: A user-friendly, extensible, and parallel library for approximate Bayesian computation. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '17. New York, NY, USA: ACM. ISBN 978-1-4503-5062-4, pp. 8:1–8:9. DOI:10.1145/3093172.3093233. URL http://doi.acm.org/10.1145/3093172.3093233.

[30] Emmerich MT, Deutz AH and Klinkenberg JW (2011) Hypervolume-based expected improvement: Monotonicity properties and exact computation. In: *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, pp. 2147–2154.

[31] Evensen G (2009) *Data Assimilation – The Ensemble Kalman Filter*. 2nd edition. Springer-Verlag Berlin Heidelberg. URL http://www.springer.com.proxy.uchicago.edu/earth+sciences+and+geography/book/978-3-642-03710-8.

[32] Fortin FA, Rainville FMD, Gardner MA, Parizeau M and Gagné C (2012) DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13: 2171–2175.

[33] Frazier PI (2018) Bayesian optimization. In: *Recent Advances in Optimization and Modeling of Contemporary Problems*. INFORMS, pp. 255–278. DOI:10.1287/educ.2018.0188. URL https://doi.org/10.1287/educ.2018.0188.

[34] Gordon N, Salmond D and Smith A (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F Radar and Signal Processing* 140(2): 107. DOI:10.1049/ip-f-2.1993.0015. URL http://digital-library.theiet.org/content/journals/10.1049/ip-f-2.1993.0015.

[35] Gramacy RB (2016) laGP: Large-scale spatial modeling via local approximate Gaussian processes in *R*. *Journal of Statistical Software* 72(1). DOI:10.18637/jss.v072.i01. URL http://www.jstatsoft.org/v72/i01/.

[36] Groen D, Arabnejad H, Jancauskas V, Edeling WN, Jansson F, Richardson RA, Lakhlili J, Veen L, Bosak B, Kopta P, Wright DW, Monnier N, Karlshoefer P, Suleimenova D, Sinclair R, Vassaux M, Nikishova A, Bieniek M, Luk OO, Kulczewski M, Raffin E, Crommelin D, Hoenen O, Coster DP, Piontek T and Coveney PV (2021) VECMAtk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379(2197): 20200221. DOI:10.1098/rsta.2020.0221. URL http://royalsocietypublishing.org/doi/10.1098/rsta.2020.0221. Publisher: Royal Society.

[37] Hartig F, Calabrese JM, Reineking B, Wiegand T and Huth A (2011) Statistical inference for stochastic simulation models –- theory and application. *Ecology Letters* 14(8): 816–827. DOI:10.1111/j.1461-0248.2011.01640.x. URL http://onlinelibrary.wiley.com.proxy.uchicago.edu/doi/10.1111/j.1461-0248.2011.01640.x/abstract.

[38] Head T, MechCoder, Louppe G, Shcherbatyi I, fcharras, Vinícius Z, cmmalone, Schröder C, nel215, Campos N, Young T, Cereda S, Fan T, rene rex, Shi KK, Schwabedal J, carlosdanielcsantos, Hvass-Labs, Pak M, SoManyUsernamesTaken, Callaway F, Estève L, Besson L, Cherti M, Pfannschmidt K, Linzberger

F, Cauet C, Gut A, Mueller A and Fabisch A (2018) scikit-optimize/scikit-optimize: v0.5.2. DOI:10.5281/zenodo.1207017. URL https://zenodo.org/record/1207017#.XbDHPSV7nxs.

[39] Holland JH (1992) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, Mass: A Bradford Book. ISBN 978-0-262-58111-0.

[40] Hunter SR, Applegate EA, Arora V, Chong B, Cooper K, Rincón-Guevara O and Vivas-Valencia C (2017) An introduction to multi-objective simulation optimization. *Optimization Online* .

[41] Jabot F, Faure T and Dumoulin N (2013) EasyABC: performing efficient approximate Bayesian computation sampling schemes using R. *Methods in Ecology and Evolution* 4(7): 684–687. DOI:10.1111/2041-210X.12050. URL http://onlinelibrary.wiley.com/doi/10.1111/2041-210X.12050/abstract.

[42] Janusevskis J, Le Riche R, Ginsbourger D and Girdziusas R (2012) Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges. In: *Learning and Intelligent Optimization*. Springer, pp. 413–418.

[43] Karypis G and Kumar V (1999) A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20(1): 359—–392.

[44] Kuhn M (2008) Building predictive models in R using the Caret package. *Journal of Statistical Software* 28(1): 1–26. DOI:10.18637/jss.v028.i05. URL https://www.jstatsoft.org/index.php/jss/article/view/v028i05.

[45] Lenormand M, Jabot F and Deffuant G (2013) Adaptive approximate Bayesian computation for complex models. *Computational Statistics* 28(6): 2777–2796. DOI:10.1007/s00180-013-0428-3. URL https://doi.org/10.1007/s00180-013-0428-3.

[46] Liaw A and Wiener M (2002) Classification and regression by RandomForest. *R News* 2(3): 18–22. URL http://CRAN.R-project.org/doc/Rnews/.

[47] Lintusaari J, Vuollekoski H, Kangasrääsiö A, Skytén K, Järvenpää M, Marttinen P, Gutmann MU, Vehtari A, Corander J and Kaski S (2018) ELFI: Engine for likelihood-free inference. *Journal of Machine Learning Research* 19(16): 1–7. URL http://jmlr.org/papers/v19/17-374.html.

[48] Luke S, Cioffi-Revilla C, Panait L, Sullivan K and Balan G (2005) MASON: A multiagent simulation environment. *SIMULATION* 81(7): 517–527. DOI:10.1177/0037549705058073. URL https://doi.org/10.1177/0037549705058073.

[49] Lusk EL, Pieper SC and Butler RM (2010) More scalability, less pain: A simple programming model and its implementation for extreme computing. *SciDAC Review* 17.

[50] Macal CM, Collier NT, Ozik J, Tatara ER and Murphy JT (2018) chiSIM: An agent-based simulation model of social interactions in a large urban area. In: *Winter Simulation Conference*.

[51] Maerzluft CE, Rutter CM, Ozik J and Collier N (2021) Incremental mixture approximate Bayesian computation (IMABC) [R package imabc version 1.0.0]. URL https://CRAN.R-project.org/package=imabc.

[52] Mahr J (2021) Is the COVID-19 pandemic growing or shrinking in Illinois? New website tracks a key metric. URL https://www.chicagotribune.com/coronavirus/ct-covid-researchers-website-reproduction-rate-20210129-ecs4cduhwrc47le5g6nm6b45fu-story.html. Section: Coronavirus, News, Breaking News.

[53] Meeds E and Welling M (2014) GPS-ABC: Gaussian process surrogate approximate Bayesian computation. In: *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*. pp. 593–602.

[54] Microsoft and Weston S (2019) *foreach: Provides Foreach Looping Construct*. URL https://CRAN.R-project.org/package=foreach. R package version 1.4.7.

[55] Moral PD, Doucet A and Jasra A (2011) An adaptive sequential Monte Carlo method for approximate Bayesian computation. *Statistics and Computing* 22(5): 1009–1020. DOI:10.1007/s11222-011-9271-y. URL http://link.springer.com/article/10.1007/s11222-011-9271-y.

[56] Morris MD (1991) Factorial sampling plans for preliminary computational experiments. *Technometrics* 33(2): 161–174. DOI:10.1080/00401706.1991.10484804. URL http://www.tandfonline.com/doi/abs/10.1080/00401706.1991.10484804.

[57] North M, Collier N, Ozik J, Tatara E, Altaweel M, Macal C, Bragen M and Sydelko P (2013) Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling* 1(3).

[58] Ozik J, Collier N and Wozniak JM (2015) Many resident task computing in support of dynamic ensemble computations. In: *Proc. MTAGS at SC*.

[59] Ozik J, Collier N, Wozniak JM and Spagnuolo C (2016) From desktop to large-scale model exploration with Swift/T. In: *Proc. Winter Simulation Conference*.

[60] Ozik J, Collier NT and Wozniak JM (2015) Many resident task computing in support of dynamic ensemble computations. In: *8th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers Proceedings*. Austin, Texas. URL http://datasys.cs.iit.edu/events/MTAGS15/program.html.

[61] Ozik J, Collier NT, Wozniak JM, Macal CM and An G (2018) Extreme-Scale Dynamic Exploration of a Distributed Agent-Based Model with the EMEWS framework. *IEEE Transactions on Computational Social Systems* 5(3): 884–895. DOI:10.1109/TCSS.2018.2859189. URL https://ieeexplore.ieee.org/document/8451972/.

[62] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M and Duchesnay E (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12: 2825–2830.

[63] Pei S, Morone F, Liljeros F, Makse H and Shaman JL (2018) Inference and control of the nosocomial transmission of methicillin-resistant Staphylococcus aureus. *eLife* 7: e40977. DOI:10.7554/eLife.40977. URL https://doi.org/10.7554/eLife.40977. Publisher: eLife Sciences Publications, Ltd.

[64] Rasmussen CE and Williams C (2006) *Gaussian Processes for Machine Learning*. MIT Press. URL http://www.gaussianprocess.org/gpml/.

[65] Richmond P and Chimeh MK (2017) FLAME GPU: Complex system simulation framework. In: *2017 International Conference on High Performance Computing Simulation (HPCS)*. pp. 11–17.

[66] Roustant O, Ginsbourger D and Deville Y (2012) DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software* 51(1): 1–55. URL https://www.jstatsoft.org/v51/i01/.

[67] Rubio-Campillo X (2014) Pandora: A versatile agent-based modelling platform for social simulation. In: *SIMUL 2014, The Sixth International Conference on Advances in System Simulation*. pp. 29–34.

[68] Rutter CM, Ozik J, DeYoreo M and Collier N (2019) Microsimulation model calibration using incremental mixture approximate Bayesian computation. *The Annals of Applied Statistics* 13(4): 2189–2212. DOI:10.1214/19-AOAS1279. URL http://projecteuclid.org/euclid.aoas/1574910041.

[69] Settles B (2012) Active Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6(1): 1–114. DOI:10.2200/S00429ED1V01Y201207AIM018. URL http://www.morganclaypool.com.proxy.uchicago.edu/doi/abs/10.2200/S00429ED1V01Y201207AIM018.

[70] Shahriari B, Swersky K, Wang Z, Adams RP and de Freitas N (2016) Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE* 104(1): 148–175.

[71] Thain D, Tannenbaum T and Livny M (2005) Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience* 17(2-4).

[72] Thomas M (2021) When COVID Came to Chicago: An Oral History. URL https://www.chicagomag.com/chicago-magazine/march-2021/67-days-to-lockdown/. Section: Chicago Magazine.

[73] Wang Z, Gehring C, Kohli P and Jegelka S (2018) Batched large-scale Bayesian optimization in high-dimensional spaces. In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 745–754.

[74] Wilensky U (1999) NetLogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. URL http://ccl.northwestern.edu/netlogo/.

[75] Wilkinson R (2014) Accelerating ABC methods using Gaussian processes. In: *Artificial Intelligence and Statistics*. PMLR, pp. 1015–1023.

[76] Wozniak JM, Armstrong TG, Wilde M, Katz DS, Lusk E and Foster IT (2013) Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. pp. 95–102. DOI:10.1109/CCGrid.2013.99.

[77] Wozniak JM, Dorier M, Ross R, Shu T, Kurc T, Tang L, Podhorszki N and Wolf M (2019) MPI jobs within MPI jobs: A practical way of enabling task-level fault-tolerance in HPC workflows. *Future Generation Computing Systems* 101. DOI:https://doi.org/10.1016/j.future.2019.05.020.

[78] Wozniak JM, Peterka T, Armstrong TG, Dinan J, Lusk EL, Wilde M and Foster IT (2013) Dataflow coordination of data-parallel tasks via MPI 3.0. In: *Proc. EuroMPI*.

[79] Zhao Y, Hategan M, Clifford B, Foster I, von Laszewski G, Raicu I, Stef-Praun T and Wilde M (2007) Swift: Fast, reliable, loosely coupled parallel computation. In: *Proc. Workshop on Scientific Workflows*.