Fast event-based epidemiological simulations on national scales

Pavol Bauer¹, Stefan Engblom¹ and Stefan Widgren²

Abstract
We present a computational modeling framework for data-driven simulations and analysis of infectious disease spread in large populations. For the purpose of efficient simulations, we devise a parallel solution algorithm targeting multi-socket shared-memory architectures. The model integrates infectious dynamics as continuous-time Markov chains and available data such as animal movements or aging are incorporated as externally defined events. To bring out parallelism and accelerate the computations, we decompose the spatial domain and optimize cross-boundary communication using dependency-aware task scheduling. Using registered livestock data at a high spatiotemporal resolution, we demonstrate that our approach not only is resilient to varying model configurations but also scales on all physical cores at realistic workloads. Finally, we show that these very features enable the solution of inverse problems on national scales.

Keywords
Computational epidemiology, discrete event simulation, multicore implementation, stochastic modeling, task-based computing

1 Introduction
Livestock diseases have a major economic impact on farmers, the livestock industry, and countries (Hasonova and Pavlik, 2006; Knight-Jones and Rushton, 2013). Modeling and simulation of infectious disease spread are important in designing cost-efficient surveillance and control (Willeberg et al., 2011). One challenge is that disease dynamics and transmission routes for various pathogens are fundamentally different. Indirect transmission of pathogens via the environment for fecal–oral diseases requires a different model compared to diseases that spread with direct contact between individuals (Brooks-Pollock et al., 2015). Another challenge is to incorporate the increasing amount of epidemiologically relevant data into the models (Pellis et al., 2015). It is therefore desirable to have simulation tools that are flexible to various disease spread models yet efficient to handle the large amounts of available livestock data.

Due to uncertainties in the exact details in pathogen transmission (Greenwood and Gordillo, 2009) and the inherent random nature of animal interactions, stochastic modeling is natural and often required. Spatial models that include proximity to infected farms with local clustering of disease spread gained popularity during the foot-mouth-disease epidemic in 2001 (Keeling, 2005; Harvey et al., 2007; Stevenson et al., 2013). Another important route for disease spread is animal trade, creating a temporal network of contacts between farms (Masuda and Holme, 2013). It has been shown that the topology and connectivity of the network have great impact on the disease spread and on the effect of control measures (Büttner et al., 2016; Shirley and Rushton, 2005).

Stochastic models on discrete state-spaces are typically simulated using discrete event simulation (DES), a general approach to evolve dynamical systems consisting of discrete events including, in particular, continuous-time Markov chains (CTMCs) (Cassandras and Lafortune, 2008). As most realistic epidemiological models are formulated on a large state-space and/or need to be studied over comparably long periods of time, parallelization is desirable. The highest degree of parallelism is typically achieved by a decomposition of the spatial information, often represented as a graph or network, into a set of subdomains (Fujimoto, 1990).

¹Division of Scientific Computing, Department of Information Technology, Uppsala University, Uppsala, Sweden
²Department of Disease Control and Epidemiology, National Veterinary Institute, Uppsala, Sweden

Corresponding author:
Stefan Engblom, Division of Scientific Computing, Department of Information Technology, Uppsala University, SE-751 05 Uppsala, Sweden. Email: stefane@it.uu.se
It is then up to the strategy for event handling at domain boundaries how well the concurrent execution scales and which overall degree of parallelism is extractable. As it may hinder scalability, a constraint that plays a crucial role in the design of parallel DES is to maintain the sequential ordering of events, that is, to preserve the underlying causality of the model.

In general, there are two types of boundary events that can occur during a simulation, which ultimately decide what will be the optimal parallelization strategy: those which are deterministic and essentially of fully predictable character and those which are stochastic and not predictable at an earlier simulation time (Fujimoto, 1999). To parallelize events that belong to the latter group, sophisticated approaches such as optimistic parallel DES algorithms have been proposed (Jefferson, 1985). These approaches may use speculative execution to enable scalability but must implement rollback mechanisms in case the event causality is violated (Carothers et al., 1999). Alternatively, in simulations where the domain crossing events are deterministic and thus predictable, conservative simulation may be used as it is possible to avoid causal violations altogether (Fujimoto, 1990; Heidelberger and Nicol, 1993). In particular, a parallel scheduler (Drozdowski, 2009) can be used to create an execution order which guarantees causality, as has been previously shown by Nicol and Liu (2002) and Xiao et al. (1999), notably with the focus on simulation of telecommunication networks.

In this article, we present an efficient and flexible framework for data-driven modeling of disease spread simulations. The model integrates disease dynamics as CTMCs and real livestock data as deterministic events. This allows us to create a temporal network of disease transmission, which has been shown to be a key aspect in modeling and simulation of spatial disease spread (Büttner et al., 2016; Shirley and Rushton, 2005). Previously, agent-based simulations based on synthetic data have been studied by others (Barrett et al., 2008; Yeom et al., 2014).

The way the model is defined allows us to predict future boundary events at any simulation time, and hence we are able to create parallel execution traces which respect causality. In particular, we find that dependency-aware task computing (Leijen et al., 2009; Subhlok et al., 1993) can be used to implement this approach with high efficiency, as all the necessary information to maintain spatial and temporal causality of events can be specified via dynamic creation of tasks and dependencies.

This is in contrast to previous approaches (Nicol and Liu, 2002; Xiao et al., 1999), as the scheduler is not an implicit part of the parallel simulation algorithm but can be chosen by the user from a wide selection of openly available libraries (e.g. OpenMP 4.0 and OmpSs, Duran et al., 2011; or StarPU, Augonnet et al., 2011). We show how the selected library is integrated into our simulation framework, by assigning parts of the sequential algorithm to independent tasks that are scheduled using a certain set of rules. We evaluate this approach using the task-parallel run-time library SuperGlue (Tillenius, 2015), which has been demonstrated to be an efficient scheduler of fine-grained tasks. Using our simulator on models with realistic workloads, we demonstrate scalability on a multi-socket shared-memory system and investigate when this approach is preferable in comparison to traditional parallelization techniques. As the achievable scalability clearly depends on the properties of the individual model, we in particular choose to investigate the influence of the model’s connectivity pattern.

The article is organized as follows. In Section 2, we introduce the mathematical foundation for our framework. In Section 3, we discuss the sequential simulation algorithm and the strategy for parallelization. In Section 4, we present numerical experiments carried out on benchmarks consisting of a recently proposed epidemiological model incorporating large amounts of registered data. We also include an example of an inverse problem for an epidemic model on national scales. Finally, in Section 5, we offer a concluding discussion around the central themes of the article.

2 Epidemiological modeling

We consider in this section a highly general approach to epidemiological modeling. Proceeding stepwise we start with a description of single-node stochastic SIR-type models in the form of CTMCs, using a compact notation that also encompasses externally defined events. We next couple an ensemble of such single-node models into a network with prescribed transitions in between the nodes to arrive at a global description. Finally, since most realistic models on multiple scales will typically incorporate also quantities for which a continuous description is more natural, we consider a mixed approach in which CTMCs are coupled to ordinary differential equations.

2.1 Discrete states

We shall use a compact notation for jump stochastic differential equations (jump SDEs) as follows. We assume a probability space \((\Omega, \mathcal{F}, \mathbb{P})\) where the filtration \(\mathcal{F}_{t\geq0}\) contains Poisson processes of any finite dimensionality. The time-dependent state vector \(X_t = X(t; \omega) \in \mathbb{Z}^N\), with \(\omega \in \Omega\), counts at time \(t\) the number of individuals of each of \(N\) different categories or compartments. Since the random process is of discrete character, the map \(t \rightarrow X(t)\) is right continuous only; by \(X(t-)\), we therefore denote the value of the state before any events scheduled at time \(t\).
Given a rate function \( r : \mathbb{Z}_+^N \rightarrow \mathbb{R}_+ \) and a stoichiometric coefficient \( s \in \mathbb{Z}_+^N \), we write a CTMC in the form:

\[
dX_t = s\mu(dt),
\]

with scalar counting measure \( \mu(dt) = \mu(r(X(t-)); dt) \). This notation expresses a dynamics consisting of events with exponentially distributed waiting times of intensities \( r(X(t-)) \); specifically \( E[\mu(dt)] = E[r(X(t-)); dt] \). An event at time \( t \) implies that the state is to be changed according to the prescription \( X(t) = X(t-) + s \). In equation (1), note that if some stoichiometric coefficient \( s_i < 0 \), then we must have that \( r(x) = 0 \) for \( x_i \) small enough, or otherwise the chain will reach negative states with positive probability.

The generalization of equation (1) to non-scalar counting measures is straightforward. Assuming \( N_i \) different transitions specified by a vector intensity \( R : \mathbb{Z}_+^N \rightarrow \mathbb{R}_+^N \) and a stoichiometric matrix \( S \in \mathbb{R}_+^{N \times N} \), we simply write:

\[
dX_t = S\mu(dt),
\]

with \( \mu(dt) = [\mu_1(dt), \ldots, \mu_N(dt)]^T \) and, for each \( k \), \( \mu_k(dt) = \mu_k(R_i(X(t-)); dt) \).

As a concrete example, consider the classical SIR model (Kermack and McKendrick, 1927):

\[
S + I \quad \frac{\beta}{\gamma} - 2I + R \bigg] .
\]

With state vector \( X = [S, I, R] \) this can be understood as:

\[
S = \begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix},
\]

\[
R(s) = [\beta x_1 x_2, \gamma x_3]^T.
\]

With one small additional convention, the above notation also encompasses events that have been defined externally. Suppose, for example, in the SIR model, that susceptible individuals are to be added one by one at known deterministic times \( (t_i) \). To accomplish this, we replace equation (4) with:

\[
S = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},
\]

and additionally define in terms of the Dirac measure:

\[
\mu_3(dt) = \sum_i \delta(t_i; dt).
\]

Equation (2) now evolves the full dynamics of the coupled stochastic–deterministic model. Note that when removing individuals using this scheme, some care is required to be able to guarantee a non-negative chain.

### 2.2 Network model

Although the previous discussion is of completely general character, it makes sense to handle the collective dynamics of a possibly very large collection of nodes in a slightly more streamlined fashion. Assuming \( N_n \) nodes in total, we consider the state matrix \( X \in \mathbb{Z}_+^{N_n \times N} \) and evolve the local dynamics by a version of equation (2):

\[
dX^{t(i)}_i = S\mu^{t(i)}(dt).
\]

Given an undirected graph \( G \) each node \( i \) is modeled to affect the state of the nodes in the connected components \( C(i) \) of \( i \), and in turn, to be affected by all nodes \( j \) such that \( i \in C(j) \). The interconnecting dynamics can then be written as:

\[
dX^{t(i)}_i = - \sum_{j \in C(i)} C\mu^{t(i)}(dt) + \sum_{j \in C(i)} C\mu^{t(i)}(dt).
\]

Note that in equation (9), global consistency is enforced as follows. The \( k \)th “outgoing” event is a change of state according to \( X^{t(k)}(t) = X^{t(k)}(t-k) + C_k \) and, for some \( j \in C(i) \), \( X^{t(k)}(t) = X^{t(k)}(t-k) + C_k \). By inspection, the intensity for this transition is \( E[x^{t(k)}(dt)] = E[N^{t(k)}(X^{t(k)}(t-k); dt)] \), say, where the dependency is only on the state of the “sending” node \( i \).

Using superposition of equations (8) and (9) the overall dynamics becomes:

\[
dX^{t(i)}_i = S\mu^{t(i)}(dt) - \sum_{j \in C(i)} C\mu^{t(i)}(dt) + \sum_{j \in C(i)} C\mu^{t(i)}(dt).
\]

As before, we conveniently allow externally defined deterministic events to be included in this description using the equivalent construction in terms of Dirac measures.

### 2.3 Continuous states

In the previous description, we assumed essentially that individuals were counted, such that a discrete stochastic model was needed to accurately capture the dynamics of a possibly small and noisy population. In a multi-scale model, however, it makes sense to allow also continuous state variables, representing, for example, environmental properties more naturally described in a macroscopic language.

Assuming an additional continuous state matrix \( Y \in \mathbb{R}_+^{N_e \times N} \) to be available, we find that a general model corresponding to equation (10) is:
\[
\frac{dY^{(i)}(t)}{dt} = f(X^{(i)}(t - ), Y^{(i)}(t)) \\
- \sum_{j \in \mathcal{C}(i)} g(X^{(i)}(t - ), Y^{(j)}(t)) + \sum_{j \in \mathcal{C}(i)} g(X^{(i)}(t - ), Y^{(j)}(t)).
\]
\[
Y_{n+1} = Y_n + \int_{t_n}^{t_{n+1}} F([X_n + X_{n+1}]/2, Y(s)) \, ds. \quad (17)
\]

In equation (16), we freeze the variable \( Y \) at a previous time step and integrate the stochastic dynamics only. Next, in equation (17), we insert an average effective value of \( X \) and integrate the deterministic part using any suitable deterministic numerical method.

To describe a more concrete numerical method, some assumptions are in order. Firstly, in equation (10), we assume that events connecting two nodes have been externally defined. In particular, this assumption is satisfied for the important case of domesticated herds of animals who move between nodes due to human interventions only. Secondly, in equation (11), we put \( g = 0 \) and thus remove all direct influence between continuous variables in connected nodes. This is reasonable for macroscopic variables that are not easily transported, like bacteria in soil, but could of course be violated for other media like groundwater or air.

For this scenario, we can write down a concrete numerical method per node \( i \) as follows:

\[
\tilde{X}^{(i)}_{n+1} = \tilde{X}^{(i)}_{n} + \int_{t_n}^{t_{n+1}} \mathbb{E} \mu^{(i)}_d(\tilde{X}^{(i)}_{n}; s, Y^{(i)}_{n}) \, ds,
\]
\[
\tilde{X}^{(i)}_{n+1} = \tilde{X}^{(i)}_{n+1} + \int_{t_n}^{t_{n+1}} \mathbb{E} \nu^{(i)}_d(\tilde{X}^{(i)}_{n}; s, Y^{(i)}_{n}) \, ds \\
- \int_{t_n}^{t_{n+1}} \mathbb{E} \nu^{(i)}_d(\tilde{X}^{(i)}_{n}; s, Y^{(i)}_{n}) \, ds \\
+ \int_{t_n}^{t_{n+1}} \mathbb{E} \nu^{(i)}_d(\tilde{X}^{(i)}_{n}; s, Y^{(i)}_{n}) \, ds,
\]

\[
Y_{n+1}^{(i)} = Y_n^{(i)} + \int_{t_n}^{t_{n+1}} F(\tilde{X}^{(i)}_{n+1}, Y^{(i)}_{n+1}) \, ds. \quad (20)
\]

In equation (18), the stochastic part (subscript \( s \)) of the measure is evolved in time to produce the temporary variable \( \tilde{X} \). Next, equation (19) incorporates all externally defined deterministic events (subscript \( d \)), both locally on the node, and according to the connectivity of the network. Finally, equation (19) is just the usual Euler forward method in time with time step \( \Delta t_n = t_{n+1} - t_n \) evolving the continuous state \( Y \). The particular splitting method (equations (18) to (20)) forms the basis for much of the results reported in Section 4.

### 3.2 External events

Similar to the epidemiological events (equation (4)), the external events modify the discrete state according to a transition vector (equation (6)), but at a predefined time \( t \).

We divide external events into two types: events of type \( E_1 \) operate on the state of a single node, while events of type \( E_2 \) operate on the states of two nodes. It
Algorithm 1: Sequential simulation loop

1. Initialize: Compute all stochastic rates $\omega_i$ in all nodes $X_i(t), i = 1, \ldots, N_n$.
2. while $t < T_{\text{end}}$ do
3. for all nodes $i = 1$ to $N_n$ do
4. while $t < (t_i + \Delta t)$ do
5. Compute the sum $\lambda$ of all transition intensity functions.
6. Sample the next stochastic event time by $\tau = -\log(\text{rand})/\lambda$ using a uniformly distributed random variable $\text{rand}$.
7. Determine which event happened. Sample the next event (by inversion); find $n$ such that
   $\sum_{j=1}^{n-1} \omega_j(X_j(t)) < \lambda \text{ rand} \leq \sum_{j=1}^{n} \omega_j(X_j(t))$.
8. Update the state $X_i(t)$ using the stoichiometric matrix $S$.
9. Update $\omega_o$, using the dependency graph $G$ to recalculate only affected stochastic rates.
10. end while
11. end for
12. $t_i + 1 = t_i + \Delta t_i$.
13. Incorporate externally defined events in lists $E_{1,2}$.
14. Loop over all nodes $X_i(t)$ and update the continuous state variable $Y_i(t)$.
15: end while

is meaningful to distinguish between these types of events, as they are processed differently by the parallel algorithm discussed later. They are defined by a set of attributes:

$$E_1 = \{ \mathbb{R}, t, n, i \},$$

$$E_2 = \{ \mathbb{R}, t, n, i, j \},$$

where $t$ is the time of the event, $\mathbb{R}$ the transition vector, $n$ the number of individuals affected, and $i$ and $j$ the indices of the affected nodes. This is a minimal set of attributes which can be further extended for specific models. As an example, within the context of the SIR model (equation (3)), we can define a birth event $\{ \mathbb{R}, t, n, i \}$ of type $E_1$ with the transition vector $\mathbb{R} = [1, 0, 0]^T$.

In the actual implementation, the transition vector is a column of the stoichiometric matrix (equation (6)) that is indexed by the event. When the event is processed at time $t$, it changes the state of node $i$ according to:

$$X_i(t)_{n+1} = X_i(t)_n + \mathbb{R}n.$$  (23)

The overall spatial domain of the model can be understood as a graph $G = (V, E)$. The edges $E$ result from events of type $E_2$ acting on source and destination nodes $X_i(t)$ and $X_j(t)$.

### 3.3 Sequential simulation algorithm

The sequential simulation algorithm is divided into three parts: the processing of stochastic events (equation (18)), hereafter referred to as the stochastic step, the processing of external events (equation (19)), or deterministic step, and the update of the continuous state variable (equation (20)). These steps are processed repeatedly in the abovementioned order until the simulation reaches the end.

The stochastic step (Algorithm 1) is an adaptation of Gillespie’s Direct Method (Gillespie, 1977). The algorithm generates a trajectory from a CTMC. At first, the rates $\omega_i(t)$ for all stochastic events $n = 1 \ldots N_n$ are evaluated in all nodes $X_i, i = 1 \ldots N_n$. Then, in each node, we sum up transition rates into $\lambda_i(t) = \sum \omega_i(t)$. Next, the algorithm uses inverse transform sampling to obtain an exponentially distributed random variable representing the next stochastic event time $t \in (0, 1)$. To obtain the index of the stochastic event that occurred within the node $X_i$, we generate a new random number $\text{rand}$ and find $n$ such that:

$$\sum_{j=1}^{n-1} \omega_j(X_j(t)) < \lambda(t) \text{ rand} \leq \sum_{j=1}^{n} \omega_j(X_j(t)).$$  (25)

When $n$ is found, we compute the state update according to the transition matrix (equation (4)), setting $X_i(t)_{n+1} = X_i(t)_n + \mathbb{R}n$ and simulation time $t = t + \tau$. Finally, to obtain the new next event time, the rate $\omega_i(t)$ of the event just occurred and all its dependent events need to be recomputed as in equation (24). For fast execution, these dependencies are stored in a dependency graph that is traversed at this stage. The algorithm repeats until a defined stopping time is reached, where the external events will next be processed.

The deterministic step works as a read and incorporate algorithm. It moves through the list of external events and processes them at the defined event time. In particular, if the event specifies a single compartment where the transition occurs, it can be directly applied to $X_i(t)$.

Finally, the continuous state variable is updated. As discussed in Section 3.1, in this step, different numerical
methods can be applied. Note that the thus updated continuous state generally affects the rate of stochastic events $\lambda^{(i)}$. Thus, before the simulation proceeds with the next iteration of the stochastic step, the event times need to be rescaled (Gibson and Bruck, 2000) using:

$$t^{(i)}_{\text{new}} = t + \left(\frac{t^{(i)}_{\text{old}}}{\lambda^{(i)}_{\text{old}}} - 1\right) \frac{\lambda^{(i)}_{\text{old}}}{\lambda^{(i)}_{\text{new}}}.$$  

(26)

The implementation of the algorithm is written in C. The overall design is inspired and partly adapted from the unstructured mesh reaction-diffusion master equation framework (Drawert et al., 2012; Engblom et al., 2009).

### 3.4 Parallel simulation algorithm

The parallelization starts with a decomposition of the spatial domain of the model understood as a graph $G = (V, E)$. The target of this graph partitioning problem is to divide the set of vertices $V$ of size $N_v$ into $k$ approximately equally sized subdomains $V_1, V_2, \ldots, V_k$. The cutting of edges $E$ follows straightforwardly from the consecutive assignment of vertices to subdomains. This partitioning strategy does not guarantee a minimum amount of edge cuts, but as the distribution of edges is predominantly homogeneous in our data, we believe that the partitioning will not benefit from more sophisticated approaches. Nonetheless, if edges are distributed heterogeneously, a minimum bisection algorithm (Andreev and Räcke, 2004) may generate an optimized cut that contributes to a better performance of the parallel solver.

After partitions $V_{1..k}$ are defined, the preprocessing algorithm continues to rearrange the external events into a structure that is more convenient for parallel processing. Firstly, the external events of type $E_1$ are assigned to $k$ lists $\mathcal{E}_1^{(i)}$, such that all $E_1$ events affecting the nodes $X^{(i)} \in V_k$ are stored in the $k$th list.

Second, external events of type $E_2$ are divided into two categories: external events of type $E_2$ where the source and destination nodes lie within the same subdomain $V_k$ are assigned to lists $\mathcal{E}_2^1$. Events in lists $\mathcal{E}_2^1$ and $\mathcal{E}_2^2$ can be processed by a thread assigned to the $k$th subdomain in private. Events of type $E_2$ where the source node and destination node do not lie within the same subdomain $V_k$ are assigned to a second list $\mathcal{E}_2^2$. This list then contains domain crossing external events that have to be handled by the simulator in a special way.

The complexity of the data rearrangement is $O(n)$, where $n$ is the number of external events. Although $n$ can be large, the workload is typically negligible for real-scale models. For example, in the national scale model presented in Section 4.1, the operation takes $\sim 0.1\%$ of the total simulation time on one core and $\sim 1\%$ of the simulation time on 32 cores, respectively. Moreover, rearranged lists can be stored and reused for models that are simulated using the same decomposition, as is done in the simulation study in Section 4.3.

Finally, the decomposed problem can be simulated in parallel. For simplicity, let us assume that each subdomain $k$ is bound to one computing thread. Then every thread processes the stochastic step (equation (18)) and the update of the continuous variable (equation (20)) on private nodes of $V_k$ as well as the deterministic step (equation (19)) on external event lists $\mathcal{E}_1^k$ and $\mathcal{E}_2^k$ (Algorithm 2). Since time has been discretized, these computations are embarrassingly parallel in which no communication between neighboring threads is necessary during the processing of the $n$th time window $t_n + \Delta t_n$. The potential bottleneck of the simulation lies in the simulation of the cross-boundary events in $\mathcal{E}_2^2$.

In our study, we handle events in $\mathcal{E}_2^2$ in different ways. The first possibility is to compute them entirely in serial. This is a valid approach if there are very few events in $\mathcal{E}_2^2$ in relation to the private events as the scaling of the private computations will not be affected. On the other hand, if the overall simulation is dominated

---

**Algorithm 2: Parallel simulation loop**

1: *Initialize*: Decompose the nodes $V$ into $k$ subdomains $V_k$. Rearrange the external events of type $E_1$ into private lists $\mathcal{E}_1^k$ of each subdomain $k$, where all $n$ affected nodes $X_n \in V_k$. Further divide all $E_2$ events into the private list $\mathcal{E}_2^k$ or the list of domain-crossing events $\mathcal{E}_2^2$.

2: while $t < T_{\text{end}}$ do

3:  for all $i = 1 \ldots k$ do

4:  % Parallel task $T_S$

5:  Execute line 14 of Algorithm 1 for all nodes in subdomain $V_i$ if $n > 1$.

6:  Execute lines 3–10 of Algorithm 1 for all nodes in subdomain $V_i$ evolving time $t \in [t_n, t_n + \Delta t]$.

7:  Execute line 13 of Algorithm 1 for all events in lists $\mathcal{E}_1^k$ and $\mathcal{E}_2^k$ at time $t \in [t_n, t_n + \Delta t]$.

8:  % End of parallel task $T_S$

9:  end for

10:  % Parallel task $T_M$

11:  Execute line 13 of Algorithm 1 for all events in the list $\mathcal{E}_2^k$ at time $t \in [t_n(t_n + \Delta t)]$.

12:  % End of parallel task $T_M$

13:  $t_{n+1} = t_n + \Delta t_n$

14: end while
by the processing of $E^c_2$ events, it can be regarded as serialized, as little concurrency will be extractable using such an approach. Hence we focus on an intermediate ratio of private and global work, where events in $E^c_2$ occur at every deterministic time step $\Delta t$ but at a lower frequency than the other private events. We will also investigate whether scaling in this regime is achievable and whether $E^c_2$ events are scheduled using dependency-aware task-computing.

### 3.5 Task-based computing

An increasing amount of scientific computations are parallelized using task-based computing (Berry et al., 2012; Haidar et al., 2014; Meng and Berzins, 2014). In order to apply this pattern, the programmer typically has to divide a larger chunk of work into a group of smaller tasks which can be processed asynchronously. A run-time library (Leijen et al., 2009; Subhlok et al., 1993) is then used to create an execution schedule of the tasks on the available parallel hardware.

If the granularity of tasks is sufficiently fine, the schedule will be denser and the idle time shorter. On the other hand, the scheduler synchronizes a larger number of small tasks which usually implies more overhead. See Gerasoulis and Yang (1993) for a thorough discussion on the impact of granularity.

If the scheduler supports dependency awareness (Perez et al., 2008), the programmer can further define a number of task dependencies. This is a critical feature if data are shared between tasks and therefore a processing order has to be enforced. The scheduler then manages the dependencies in the form of a directed acyclic graph and spawns tasks whenever all dependencies are met.

We believe that the usage of task-based computing is beneficial in our computational framework, as a small granularity of processes is given by the underlying modeling. In our approach, we aim to divide our computations into tasks and define a scheduling policy which guarantees causality of events although they are processed in parallel.

These scheduling rules can be implemented on any dependency-aware task scheduler, the only requirement for some of the scheduling policies is the support for dynamic addressing of a subset of dependencies, for example, via an array of pointers. For example, OpenMP 4.0 does not support this (OpenMP Architecture Review Board, 2013). In our computational experiments, we make use of the run-time library SuperGlue (Tillenius, 2015). In SuperGlue, dependencies are assigned to data and expressed via data versioning (Zafari et al., 2012). If a chunk of data is being processed by a task, a version counter representing the data access will be increased. Other tasks that are dependent on the chunk will be spawned whenever the new version becomes available. SuperGlue has been demonstrated to be an efficient shared-memory task scheduler that it is capable of operating at a comparably low synchronization overhead. The processing of dependencies and spawning of tasks is dynamic, and SuperGlue additionally supports load balancing by work stealing from over-utilized threads.

### 3.6 Scheduling and dependencies

We now define the tasks and their dependencies that are used in the task-based implementation of the parallel algorithm. Task $T_S(k, n)$ executes private computations on the decomposed data of the $k$th subdomain (lines 5 to 7 of Algorithm 2). That is the stochastic step (equation (18)) on all nodes $V_k$, the processing of the private external events in lists $E^c_1$ and $E^c_2$ (equation (19)) as well as the update of the continuous variables (equation (20)). The counter $n$ indicates the iteration of the time window $t_n+1 = t_n + \Delta t_n$.

Task $T_M$ processes state updates due to the domain crossing events stored in list $E^c_2$. In order to estimate the possible impact of granularity of $T_M$ tasks, we compare two different scheduling policies; if the task is constructed for coarse-grained processing, we compute all $E^c_2$ events occurring at the $n$th time window $\Delta t$ in one single task. Thus, the task takes only one argument, $T_M(n)$.

If tasks are constructed for fine-grained processing, we schedule each event in $E^c_2$ as a distinct task. We then denote the task by $T_M(k_1, k_2, i)$, where $k_1$ and $k_2$ are the subdomains subject to an $E^c_2$-event update, in which two nodes $X^{(m)} \in V_1$ and $X^{(n)} \in V_2$ are affected. The counter $i$ now denotes the total order of the $E^c_2$ events in $E^c_2$ as given by the model input. This implies that if $1 \ldots n$ events exist in the window $[t_{n}, t_{n+1}]$, they have to be processed by the task in this order.

Both tasks $T_S$ and $T_M$ are scheduled repeatedly until the simulation reaches its end time. Precedence dependencies between tasks are expressed using the “$<$” operator. For example, $T_S(k_1, n) < T_S(k_1, m)$ means that task $T_S(k_1, n)$ must complete its execution before task $T_S(k_1, m)$ is spawned. Our task-based implementation contains the following dependencies:

1. $T_S(k, n) < T_S(k, m)$ if $n < m$, to maintain the causality of private updates of subdomain $V_k$.
2. To maintain the causality of domain crossing events:
   (a) $T_M(n) < T_M(m)$ if $n < m$, at coarse-grained processing.
   (b) $T_M(k_a, k_b, n) < T_M(k_c, k_d, m)$ if $n < m$ and $k_c \in \{k_a, k_b\}$ or $k_d \in \{k_a, k_b\}$, at fine-grained processing.
3. To maintain the causality between private subdomain updates and domain-crossing events:
utilization of the task scheduler. Firstly, the task
on the bookkeeping overhead. potentially be larger. This can clearly have an impact
met. In the coarse-grained case, the number of depen-
T
created for each single task
VTEC O157 dynamics, using the presented framework.
spread mechanisms, we have created a model of the
bacteria require a thorough understanding of the source
out any signs of clinical disease (Hancock et al., 2001).
infected with VTEC O157 are an important reservoir for
cause severe disease in humans, notably children
is a zoonotic bacterial pathogen with the potential to
spread
4.1 National scale simulation of VTEC bacteria
simulation of VTEC bacteria
4 Computational experiments
In the following section, we present results of computa-
tional experiments of our simulator. The following
measurements were obtained on Sandy: a Dell Power
Edge R820 computer system equipped with four Intel
Xeon E5-4650 processors and eight cores on each
socket. We restricted the execution to available physical
cores, as timing results on hyper-threads were strongly
fluctuating. We begin with a real-world simulation
using animal movement data on national scales, fol-
lowed by a synthetic benchmark for scalability at
varying connectivity load, and we conclude with a
compute-intensive parameter estimation example.

4.1 National scale simulation of VTEC bacteria
spread
Verotoxigenic Escherichia coli O157: H7 (VTEC O157)
spread
is a zoonotic bacterial pathogen with the potential to
cause severe disease in humans, notably children
(Karmali et al., 1983a, 1983b; Riley et al., 1983). Cattle
infected with VTEC O157 are an important reservoir for
the bacteria and they shed the bacteria in the feces with-
out any signs of clinical disease (Hancock et al., 2001).
Reducing the prevalence of infected cattle in the popula-
tion could potentially reduce the number of human
cases. However, the epidemiology of VTEC O157 in catt-
le is complex and targeted interventions to control the
bacteria require a thorough understanding of the source
and transmission routes (Hancock et al., 2001).

To explore the feasibility of national scale simulations
to improve the understanding of the underlying disease
spread mechanisms, we have created a model of the
VTEC O157 dynamics, using the presented framework.
European Union legislation requires member states to
keep register of bovine animals including the location
and the date of birth, movements between holdings, and
date of death or slaughter (Anonymous, 2000, 2004).
These records enable data-driven disease spread simu-
lations that include spatiotemporal dynamics of the cattle
population with regard to age structures, births, herd
size, slaughter, and trade patterns.

The present computational experiment is based on
all cattle reports to the Swedish Board of Agriculture
over the period July 1, 2005 to 31 December, 2013.
From these reports, three types of $E_1$ external events
(enter, exit, and aging) and a single type of $E_2$ event
(animal movements) were condensed. In total, there
were $\sim 10^8$ external events processed during the total
time run of $T = 3106$ days. We let each integer in
0, 1, \ldots, $T$ to represent a synchronization window for
external events, where in each window 3707+670 $E_1$
events and 235+104 $E_2$ events were processed. A subset
of the spatial network consisting of $N_n = 37221$ nodes
is visualized in Figure 1.

Most infected cattle shed the bacteria less than 30
days before returning to the susceptible state, but calves
shed for a longer period than adult cattle (Cray and
Moon, 1995; Davis et al., 2006). To capture this, we let
the intensity of the transitions between the states
depend on the $\delta$th age category:

$$S_j \xrightarrow{\eta_j} I_j, \quad I_j \xrightarrow{\gamma_j} S_j.$$ (27)

The rate for a susceptible individual on the $i$th node
to become infected per unit of time is given by:

$$\eta_j = u \phi_i(t), \quad \gamma_j = \frac{u}{\delta_j},$$ (28)

for $i = 1, \ldots, N_n$ and $j \in \{\text{calves, youngstock, adults}\}$. In turn, the expected time an infected individual is in an
infected state before it returns to the susceptible state is:

$$\eta_j$$

where $\delta = [28, 25, 22]$ and $\nu = [8, 7, 1] \times 10^{-3}$ are
age-dependent constants. The factor $u$ can be under-
stood as a time scale and is difficult to estimate accu-
rately; in our experiments, it is in fact varied such that
$u = 1$ closely resembles the parameterization of the
model found in Widgren et al.’s (2016) study.

Finally, the continuous variable $\phi_i$ represents the
environmental bacterial concentration that asserts an
infectious pressure on each individual at the $i$th node.
A suitable model is given by:

$$\frac{d\phi_i}{dt} = \alpha \sum_j I_{i,j}(t) - \beta(t)\phi_i(t).$$ (30)
Again, \( i = 1, \ldots, N_n \) are the nodes and \( S_{i,j} \) and \( I_{i,j} \) refer to the number of susceptible and infected individuals in the \( j \)th age compartment at the \( i \)th node, respectively. The constant \( \alpha \) is the average shedding rate of bacteria to the environment per infected individual, while \( \beta \) captures the decay and removal of bacteria. In our experiments, we used the constant value \( \alpha = 1 \) while \( \beta(t) \) varied according to the season:

\[
\beta(t) = \begin{cases} 
\frac{\log(2)}{14} : & 0 \leq (t \mod 365) \leq 91 \\
\frac{\log(2)}{26} : & 91 < (t \mod 365) \leq 182 \\
\frac{\log(2)}{20} : & 182 < (t \mod 365) \leq 273 \\
\frac{\log(2)}{12} : & 273 < (t \mod 365) \leq 364 
\end{cases}
\]  

We first parallelized the simulation by spreading tasks \( T_S \) over multiple cores using OpenMP and serially processing the intermediate \( E_c \) events, hereafter referred to as the fork-join approach. Next, we simulated the model using the task-based approach, scheduling tasks with coarse-grained and fine-grained policies as described in Section 3.6. We chose the number of subdomains \( k \) to be a multiple of the number of threads \( c \). Note that this is also the number of tasks \( T_S \) scheduled for each time window \( \Delta t \). As a higher factor \( u \) creates a higher load for the tasks \( T_S \), we vary \( u \) to inspect boundary regions of the parallel performance.

The scaling of the different approaches is shown in Figure 2. For the case of \( u = 1 \), we find that task sizes are too small to be efficient in both task-based approaches, and thus the fork-join approach reaches a higher efficiency.

At \( u = 10 \), we observe that coarse-grained processing performs better than the fork-join parallelization, optimally at task sizes \( k = 16c \) and \( k = 32c \). In the case of fine-grained task processing, we found that the choice of \( k \) has a strong impact on the performance scaling. While all task densities scale strongly at a lower thread count, only the \( k = 16c \) density reaches a high efficiency of 0.58 at full thread consumption. Thus the efficiency is more than doubled in comparison to the efficiency of the fork-join parallelization, which was found to be 0.23.

The dependency of the parallel efficiency on the factor \( u \) is further detailed in Figure 3. We observe that scheduling overhead and small task sizes prohibit a high efficiency of both task-based approaches if \( u < 1 \) while the full potential of the approaches is extractable at \( u > 1 \) and a larger thread count. Note that the thread affinity of tasks was varied throughout the performed experiments in order to investigate the impact of data locality.

We further present a set of characteristics of the coarse-grained and fine-grained simulations in Tables 1 and 2. As shown in Table 1, the granularity of the fine-grained task \( T_M \) is \(~1/30\) of the granularity of the coarse-grained task \( T_M \), however the task needs to be scheduled about 235 times more often throughout the simulation run.

On the other hand, the advantage of the fine-grained scheduling is emphasized by the measurements shown in Table 2; the average waiting time to fulfill the dependencies for the fine-grained \( T_M \) is 44–108 times less than for the coarse-grained \( T_M \). This is explained by the larger number of dependencies associated to the coarse-grained \( T_M \) which is growing with the partitioning \( k \).

---

**Figure 1.** Visualization of cattle movements in the VTEC O157 disease spread simulation (Section 4.1). The arcs shown are a random subset of the complete dataset of \(~10^8\) recorded events. The source of the data is the national cattle register at the Swedish Board of Agriculture. VTEC O157: verotoxigenic *Escherichia coli* O157: H7.
The resulting execution trace is also visualized in Figure 4, where we show that fine-grained $T_M$ tasks interleave more densely with tasks $T_S$, thus leading to lower idle times and higher parallel efficiency. The percentage of total work spent on the processing of tasks, the synchronization of worker threads, as well as the time spent waiting for fulfilled dependencies are shown in Figure 5 for the $u = 10$ configuration.

For further details of the scheduling performance of the SuperGlue library in regard to the task sizes and the number of dependencies, we like to refer the reader to the benchmarks available in the study by Tillenius (2015).

### 4.2 Synthetic benchmark

The results in Section 4.1 indicate a delicate performance dependency on the balance between the local events and the effective connectivity of the network. To further investigate this, a synthetic benchmark with a fixed load of local events was created. This model

![Figure 2. Performance measurements of the VTEC O157 model simulation on Sandy at varying scheduling approaches, task sizes, and scale factor $u$. The number of tasks $k$ is chosen to be proportional to the number of threads $c$. In the OpenMP parallelization ("fork–join"), cross-boundary events are processed entirely in serial. Error bars represent the standard error in mean ($n = 10$). VTEC O157: verotoxigenic Escherichia coli O157: H7.](image)

**Table 1.** Average task granularity ± the standard deviation and the total number of tasks created during the simulation at a given partitioning $k$.

| $k$   | $T_S$  | $T_M$ coarse | $T_M$ fine  | $T_S$  | $T_M$ coarse | $T_M$ fine  |
|-------|--------|--------------|-------------|--------|--------------|-------------|
| 256   | 596 ± 799 | 314 ± 132    | 11 ± 5.2    | 794112 | 3102         | 731889      |
| 512   | 301 ± 461 | 328 ± 145    | 11 ± 5.2    | 1588224 | 3102         | 731889      |
| 1024  | 152 ± 294 | 327 ± 145    | 11 ± 5.2    | 3176448 | 3102         | 731889      |
consists of two compartments $S$ and $I$ only, both residing on $N_n = 1000$ nodes. The transitions are simply:

$$
\begin{align*}
S &\xrightarrow{1} I, \\
I &\xrightarrow{1} S,
\end{align*}
$$

where the initial population size of each compartment $I_i$ and $S_i$ was set to 1000. This model is considered at times $t = [0, \Delta t, 2\Delta t, \ldots T]$, with $\Delta t = 1$ and $T = 1000$, thus generating about 2000 local events per synchronization time window $\Delta t$ and node $i$.

The nodes were arranged into $k$ subdomains and a total of $\rho k(k-1)/2$ distinct $E_2$ events were generated at the end of each time window, each connecting two randomly sampled nodes $X^{(i)}$ and $X^{(j)}$ belonging to different subdomains. Hence $\rho = 1$ means that all subdomains

Table 2. Maximum $\pm$ full width at half maximum of the (right-skewed) histogram of waiting times for fulfilled task dependencies, and the average amount $\pm$ standard deviation of dependencies assigned to tasks at each discrete time interval $[t_n, t_{n+1}]$, at a given partitioning $k$ on 32 computing cores.

| $k$ | $T_s$ coarse | $T_M$ coarse | $T_s$ fine | $T_M$ fine | $T_s$ | $T_M$ coarse | $T_M$ fine |
|-----|---------------|---------------|------------|------------|-------|---------------|------------|
| 256 | 15 ± 10       | 550 ± 200     | 12.5 ± 5   | 12.5 ± 5   | 1     | 146 ± 33     | 2          |
| 512 | 12.5 ± 5      | 850 ± 200     | 12.5 ± 5   | 12.5 ± 5   | 1     | 202 ± 58     | 2          |
| 1024| 12.5 ± 5      | 1350 ± 500    | 11 ± 4     | 11.5 ± 4   | 1     | 248 ± 83     | 2          |
have to communicate with all other subdomains at each synchronization point. The number of tasks for the coarse-grained and fine-grained approach was set to the number of threads \( k = c \).

The measurements obtained on the Sandy computer system at full thread consumption are presented in Figure 6. The parallel efficiency of all methods lies at \( 0 - 7\) for \( r/C_2 0 \) and remains there even when \( r \to 0 \) and so we deduce that the problem is memory bound. The coarse-grained task-based implementation and the fork–join approach scale very similarly with increasing connectivity \( r \). The fine-grained task-based approach attains the highest parallel efficiency at \( r/C_2 0 \), but the performance drops at a higher global connectivity. This phenomenon arises because each \( T_M \) task creates dependencies on two subsequent \( T_S \) tasks at every synchronization window, thus creating a higher overhead and limiting asynchronous task execution.

### 4.3 Feasibility of parameter estimation

A usually very compute intensive load case is the fitting of model parameters, typically using numerical optimization of some kind. The problem can briefly and ideally be described as follows; unknown is the set of parameters \( k^* \) and an observed time-series of data \( X(t; k^*) \). The parameters \( k^* \) are estimated by repeatedly simulating a whole family of trajectories with parameters \( k \), where \( k \) is modified until input data and simulations match up in some suitable sense. The framework’s feasibility of fitting an epidemiological model is of course very important, since the modeling process at some point or the other will involve in calibration of parameters with respect to reference data.

To demonstrate the feasibility of parameter estimation in the current context, we use the epidemiological model introduced in Section 4.1 and first identify the set of parameters \( [k_1, k_2, k_3] \) that have a high degree of observability. A suitable such set is:

\[
k_j = v_j \delta_j, j \in \{\text{calves, young stock, adults}\}.
\]
We let a reference solution be given by a single trajectory \( \bar{x}(t; k^*) \), with \( \delta^* = [28, 25, 22], \nu = [8.8, 3.2, 1] \times 10^{-3} \), and with \( \alpha \) and \( \beta = \beta(t) \) as in Section 4.1.

To obtain a robust procedure, some kind of smoothing statistics should be considered. We chose to aggregate counts of animals in neighboring nodes into larger regions. To be precise, the overall domain was divided into 21 areas (coinciding with the Swedish county codes), after which the goodness of fit \( G(k) \) was defined by:

\[
G(k)^2 = \int_0^T \left( \sum_j \bar{x}_j(t; k) - \sum_{i \in C(j)} \bar{x}_i(t; k^*) \right)^2 dt,
\]

with:

\[
\bar{x}_j(t; k) = \frac{1}{N} \sum_{i=1}^N \bar{x}_i(t; k),
\]

and where the individual sample trajectories are given by:

\[
\bar{x}_i(t; k) = \sum_{i \in C(j)} \bar{x}_i(t; k, \omega_i),
\]

where \( N \) is the number of trajectories and \( C(j), j \in \{1, \ldots, 21\} \) is the set of nodes \( \bar{x}_i(t) \) that belongs to county \( j \). To quantify the uncertainty in \( G(k) \), we compute the variance as:

\[
VG(k)^2 = \int_0^T \sum_j \sigma_j^2(t; k) dt,
\]

where:

\[
\sigma_j^2(t; k) = \frac{1}{N-1} \sum \| \bar{x}_j(t; k) - \bar{x}_j(t; k) \|^2,
\]

and use \( \pm 2VG(k)/\sqrt{N} \) as a measure of the uncertainty.

Next, the parameter estimation problem is approached by solving the minimization problem:

\[
\hat{k} = \arg \min_k G(k)^2.
\]

In practice, we make use of the pattern search routine in (Hooke and Jeeves, 1961), which conceptually resembles the golden section search (Kiefer, 1953) in its narrowing of the search space. The numerical optimization routine evaluates (equation (34)) until the residual error reaches a defined threshold. In our tests, we varied the initial guess of the parameters \( k_0 \) but found that the results did not vary substantially. In the results below, we conveniently put \( k_0 = 1.6k^* \).

Since an increasing number of trajectories yields better estimates of the mean and variance, we simulate using different number of trajectories. We measure the total solver time on 12 and 32 computing cores, respectively, and we let the total number of iterations to be \( N = 20 \) in all cases. The results are presented in Table 3, where the relative residual is defined as:

\[
R(k) = \frac{|G(k) - G(k^*)|}{|G(k^*)|}.
\]

The optimization landscape of the goal function (equation (4.8)), and hence the definiteness of the setup itself, is visualized in Figure 7. Due to the simple bisection search behavior of the numerical routine, the obtained parameters \( k \) are in fact the same for all displayed cases, although the relative residuals differ considerably.

Note that the obvious approach of parallelization by computing the \( N \) independent trajectories using separate threads by a sequential algorithm is unfavorable here, for two related reasons. Firstly, each executable needs to store a rather large state space in working memory. Secondly, each simulation must also access the complete database of externally scheduled events.

### 5 Conclusions

Modeling and simulation are important in designing surveillance and control of livestock diseases and of major economic importance. However, various
pathogens require different models to capture the disease dynamics and transmission routes. Moreover, an increasing amount of epidemiologically relevant data is becoming available. We have addressed these challenges and present a flexible and efficient computational framework for modeling and simulation of disease spread on a national scale. The simulation involves two parts. Firstly, the algorithm evolves stochastic dynamics of the disease process. Secondly, a list processor incorporates database events such as entering, exit, or movement of individuals into the model state. The framework is highly flexible in that most conceivable epidemiological models are either directly expressible, or the framework may be straightforwardly extended to encompass also non-standard models. As a concrete example, it would be relatively easy to include intervention strategies such as vaccination programs in order to simulate the impact on the global dynamics.

We have explored different strategies to parallelize the simulator on multi-socket architectures. Firstly, we decomposed the spatial information and the list of deterministic events. We then observed that the decomposed problem can be simulated at a high parallel efficiency, which is limited only by the processing of cross-boundary events. We then created three parallel implementations of the simulator core; we used OpenMP to only parallelize private computations in a fork–join fashion, while cross-boundary events were processed in serial. Two further implementations use a dependency-aware task scheduler to create execution traces that interleave cross-boundary events and private computations with respect to their dependencies. We find that this strategy allows us to exploit shared-memory parallelism at a higher degree than the fork–join approach if task sizes are sufficiently large. We benchmark this approach using the SuperGlue task library but present a set of scheduling rules defining the parallel simulator on general terms, thus allowing it to be implemented also with other dependency-aware task libraries.

We benchmarked our simulator using a model of the spatiotemporal spread of VTEC O157 bacteria in the Swedish cattle population. The model contains 37,221 nodes and evolves $\sim 10^6$ external events from register data. We found that at a low private workload, the fork–join approach performs best, mainly due to the scheduling overhead of the task-based approaches. For higher private workloads, the simulation benefits from task-based computing, doubling the parallel efficiency at a higher degree than the fork–join approach if task sizes are sufficiently large. We benchmark this approach using the SuperGlue task library but present a set of scheduling rules defining the parallel simulator on general terms, thus allowing it to be implemented also with other dependency-aware task libraries.

In a final example, we used the simulator to carry out an experimental parameter fitting within the VTEC O157 bacteria spread model. We emphasize the high computational complexity of this task with multiple unknown parameters to fit and the need to use several full simulation runs to evaluate each parameter candidate. A similar load case results when different intervention strategies are to be evaluated. For example, even when several interventions reduce the infectious spread globally, a policy maker could be interested in finding the most cost-efficient strategy. With this work, we provide a powerful, highly general and freely available software, which can contribute to a rapid and more efficient development of realistic large-scale epidemiological models.

Future research will encompass studies of larger inverse problems, including more realistic data input, and more complex dynamics. Yet another point for future study is the scalability of the task-based approach in a distributed environment.

Acknowledgements

We thank Martin Tillenius for providing assistance with the use of SuperGlue.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was financially supported by the Swedish Research Council within the UPMARC Linnaeus center of Excellence (P. Bauer, S. Engblom).

References

Andreev K and Räcke H (2004) Balanced graph partitioning. In: Proceedings of the 16th ACM SPAA, SPAA ’04. New York, NY, USA, 27–30 June 2004, pp. 120–124. New York: ACM.

Anonymous (2000) Regulation (EC) No 1760/2000 of the European Parliament and of the Council of 17 July 2000 establishing a system for the identification and registration of bovine animals and regarding the labelling of beef and beef products. Official Journal of the European Union L 204: 1–10.

Anonymous (2004) Commission Regulation (EC) No 911/2004 of 29 April 2004 implementing Regulation (EC) No 1760/2000 of the European Parliament and of the Council as regards eartags, passports and holding registers. Official Journal of the European Union L 163: 65–70.
Augonnet C, Thibault S, Namyst R, et al. (2011) StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation Practice and Experience* 23(2): 187–198.

Barrett C, Bisset K, Eubank S, et al. (2008) EpiSimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In: *Proceedings of Supercomputing 2008*, Austin, TX, 15–21 November 2008, pp. 1–12. Piscataway, NJ: IEEE.

Berry LA, Elwasif W, Reynolds-Barreto JM, et al. (2012) Event-based parareal: a data-flow based implementation of parareal. *Journal of Computational Physics* 231(17): 5945–5954.

Brooks-Pollock E, de Jong M, Keeling M, et al. (2015) Eight challenges in modelling infectious livestock diseases. *Epidemiology* 10: 1–5.

Böttner K, Krieter J, Traulsen A, et al. (2016) Epidemic spreading in an animal trade network - comparison of distance-based and network-based control measures. *Transboundary and Emerging Diseases* 63: e122–e134.

Carothers CD, Perumalla KS and Fujimoto RM (1999) Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulation* 9(3): 224–253.

Cassandras CG and LaFortune S (2008) Systems and models. In: *Introduction to Discrete Event Systems*. Boston, MA: Springer, pp. 1–51.

Cray WC and Moon HW (1995) Experimental infection of calves and adult cattle with Escherichia coli O157: H7. *Applied and Environmental Microbiology* 61(4): 1586–1590.

Davis MA, Rice DH, Sheng H, et al. (2006) Comparison of cultures from rectoanal-junction mucosal swabs and feces for detection of Escherichia coli O157 in dairy heifers. *Applied and Environmental Microbiology* 72(5): 3766–3770.

Drawert B, Engblom S and Hellander A (2012) URDME: a modular framework for stochastic simulation of reaction-transport processes in complex geometries. *BMC Systems Biology* 6(1): 76.

Drozdowski M (2009) Parallel tasks. In: Sammes AJ (ed.) *Scheduling for Parallel Processing (Computer Communications and Networks).* London: Springer, pp. 87–208.

Duran A, Ayguade E, Badia RM, et al. (2011) OmpSs: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters* 21(02): 173–193.

Engblom S (2015) Strong convergence for split-step methods. *SIAM Journal on Numerical Analysis* 53(6): 2655–2676.

Engblom S, Ferm L, Hellander A, et al. (2009) Simulation of stochastic reaction-diffusion processes on unstructured meshes. *SIAM Journal on Scientific Computing* 31(3): 1774–1797.

Fujimoto RM (1990) Parallel discrete event simulation. *Communications of the ACM* 33(10): 30–53.

Fujimoto RM (1999) *Parallel and Distribution Simulation Systems*. 1st ed. New York: John Wiley & Sons, Inc.

Gerasoulis A and Yang T (1993) On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems* 4(6): 686–701.

Gibson MA and Bruck J (2000) Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A* 104(9): 1876–1889.

Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry* 81(25): 2340–2361.

Greenwood PE and Gordillo LF (2009) Stochastic epidemic modeling. In: Chowell G, Hyman JM, Bettencourt LMA, et al. (eds) *Mathematical and Statistical Estimation Approaches in Epidemiology*. Amsterdam, Netherlands: Springer, pp. 31–52.

Haidar A, Tomov S, Dongarra J, et al. (2014) A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine-grained memory aware tasks. *International Journal of High Performance Computing Applications* 28(2): 196–209.

Hancock D, Besser T, Lejeune J, et al. (2001) The control of VTEC in the animal reservoir. *International Journal of Food Microbiology* 66(1-2): 71–78.

Harvey N, Reeves A, Schoenbaum MA, et al. (2007) The North American animal disease spread model: a simulation model to assist decision making in evaluating animal disease incursions. *Preventive Veterinary Medicine* 82(3-4): 176–197.

Hasonova L and Pavlik I (2006) Economic impact of paratuberculosis in dairy cattle herds: a review. *Veterinarni Medicina-Czech* 51(5): 193–211.

Heidelberger P and Nicol D (1993) Conservative parallel simulation of continuous time Markov chains using uniformization. *IEEE Transactions on Parallel and Distributed Systems* 4(8): 906–921.

Hooke R and Jeeves TA (1961) Direct search: solution of numerical and statistical problems. *Journal of the ACM* 8(2): 212–229.

Jefferson DR (1985) Virtual time. *ACM Transactions on Programming Languages and Systems* 7(3): 404–425.

Karmali MA, Petric M, Lim C, et al. (1983a) Escherichia coli cytotoxin, haemolytic-uraemic syndrome, and haemorrhagic colitis. *Lancet* 2(8362): 1299–1300.

Karmali MA, Steele BT, Petric M, et al. (1983b) Sporadic cases of haemolytic-uraemic syndrome associated with faecal cytotoxin and cytotoxin-producing escherichia coli in stools. *Lancet* 1(8325): 619–620.

Keeling MJ (2005) Models of foot-and-mouth disease. *Proceedings of the Royal Society B* 272(1569): 1195–1202.

Kermack WO and McKendrick AG (1927) A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society A* 115: 700–721.

Kiefer J (1953) Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society* 4(3): 502–506.

Knight-Jones TJD and Rushton J (2013) The economic impacts of foot and mouth disease – what are they, how big are they and where do they occur? *Preventive Veterinary Medicine* 112(3-4): 161–173.

Leijen D, Schulte W and Burckhardt S (2009) The design of a task parallel library. In: *Proceedings of the 24th ACM SIGPLAN Conference on object oriented programming systems languages and applications, OOPSLA*. New York, NY, USA, pp. 227–242. New York: ACM.

Masuda N and Holme P (2013) Predicting and controlling infectious disease epidemics using temporal networks. *F1000Prime Reports* 5: 6.

Meng Q and Berzins M (2014) Scalable large-scale fluid–structure interaction solvers in the uintah framework via hybrid task-based parallelism algorithms. *Concurrency and Computation Practice and Experience* 26(7): 1388–1407.
Nicol D and Liu J (2002) Composite synchronization in parallel discrete-event simulation. *IEEE Transactions on Parallel and Distributed Systems* 13(5): 433–446.

OpenMP Architecture Review Board (2013) *OpenMP 4.0 Application Program Interface*, p. 117, Lines 14–15.

Pellis L, Ball F, Bansal S, et al. (2015) Eight challenges for network epidemic models. *Epidemics* 10: 58–62.

Perez J, Badia R and Labarta J (2008) A dependency-aware task-based programming environment for multi-core architectures. In: *2008 IEEE International Conference on Cluster Computing*, Tsukuba, 29 September–1 October 2008, pp. 142–151. IEEE.

Riley LW, Remis RS, Helgerson SD, et al. (1983) Hemorrhagic colitis associated with a rare Escherichia coli serotype. *New England Journal of Medicine* 308(12): 681–685.

Shirley MD and Rushton SP (2005) The impacts of network topology on disease spread. *Ecological Complexity* 2(3): 287–299.

Stevenson MA, Sanson RL, Stern MW, et al. (2013) InterSpread plus: a spatial and stochastic simulation model of disease in animal populations. *Preventive Veterinary Medicine* 109(1–2): 10–24.

Subhlok J, Stichnoth JM, O’Hallaron DR, et al. (1993) Exploiting task and data parallelism on a multicomputer. In: *Proceedings of the 4th PPOPP*, New York, NY, USA: ACM, pp. 13–22.

Tillenius M (2015) SuperGlue: a shared memory framework using data versioning for dependency-aware task-based parallelisation. *SIAM Journal on Scientific Computing* 37(6): C617–C642.

Widgren S, Engblom S, Bauer P, et al. (2016) Data-driven network modeling of VTEC O157 transmission in Swedish cattle using complete population movement data. Under revision.

Willeberg P, Grubbe T, Weber S, et al. (2011) The world organisation for animal health and epidemiological modelling: background and objectives. *Revue Scientifique et Technique* 30(2): 391–405.

Xiao Z, Unger B, Simmonds R, et al. (1999) Scheduling critical channels in conservative parallel discrete event simulation. In: *Proceedings of the 13th Workshop of PADS*, Atlanta, GA, 1–4 May, 1999, pp. 20–28. Washington, DC: IEEE.

Yeom JS, Bhaete A, Bisset K, et al. (2014) Overcoming the scalability challenges of epidemic simulations on Blue Waters. In: *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium*, Phoenix, AZ, pp. 755–764.

Zafari A, Tillenius M and Larsson E (2012) Programming models based on data versioning for dependency-aware task-based parallelisation. In: *Proceedings of the 15th International Conference on Computational Science and Engineering*, Nicosia, Cyprus, 5–7 December 2012, pp. 275–280. Los Alamitos: IEEE Computer Society.

**Author biographies**

**Pavol Bauer** is a PhD candidate at the Department of Information Technology, Uppsala University. He is associated with the Linnaeus center of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center. His research interests include scientific computing, high-performance computing, and modeling and simulation with applications in Biomedicine in general. He received his master degree in Computer Science from Vienna University of Technology in 2011.

**Stefan Engblom** is an associate professor at the Department of Information Technology, Uppsala University and is associated with the Linnaeus center of excellence UPMARC. His research is centered around scientific computing and numerical analysis with a focus in the biosciences. He received his PhD from the Uppsala University in 2008 and became a Docent there in 2013.

**Stefan Widgren** is a veterinary epidemiologist at the Swedish National Veterinary Institute and a PhD candidate at the Swedish University of Agricultural Sciences. His work is focused on animal disease surveillance as well as research on risk factors for introduction and spread of VTEC O157:H7 in Swedish cattle herds.