Accelerating Simulation for Agent-Based Epidemic Models using FPGAs

Ce Guo† Wayne Luk† Stephen Weston†‡
† *Department of Computing, Imperial College London, UK*
‡ *Risk Advisory Group, Deloitte, UK*
† {c.guo, w.luk}@imperial.ac.uk ‡ sweston@deloitte.co.uk

***Abstract*—Agent-based models (ABMs) play a critical role in the containment and mitigation of epidemics. An ABM for epidemics involves a population of agents that represent healthy and infected individuals. Users can simulate the interactions of the agents to analyse spreading patterns, make predictions and conduct what-if tests. A major drawback that limits the application of ABMs is the long simulation time for large populations. This paper proposes new techniques to speed up ABM-based simulation. Specifically, we present an agent-based model for epidemic spreading that enables effective acceleration using reconfigurable hardware. The key idea is to compute the infection probability in a novel way so that the amount of on-chip memory usage is independent of the population size. Also, we propose a parameter pre-computation algorithm, a parallel simulation algorithm and an efficient collaboration method between the host computer and the accelerator. We simulate the proposed model on an Intel Arria 10 FPGA and compare it with a software reference that simulates a conventional ABM on an Intel Core i5-9400F CPU with six cores. The two systems produce similar results, but the FPGA-based system achieves 14 times speedup compared to the software reference.**

***Index Terms*—agent-based modelling, epidemics simulation, reconfigurable computing, limited-memory algorithms, clustering**

I. Introduction

An epidemic is the rapid spreading of an infectious disease that affects a large population or region [1], [2]. Even in the modern world, epidemics can still lead to a public health crisis and economic recession. For instance, the recent global outbreak of COVID-19 has claimed the lives of over 832000 people and has plunged the global economy into recession all within a few short weeks. In order to mitigate an epidemic, it is necessary to understand how it spreads.

Agent-based models (ABMs) are useful tools to analyse the spread of epidemics. An ABM treats each individual as an agent in a simulated world. The simulation algorithm of the model tracks the health status of each agent. ABMs can capture realistic dynamics of the epidemic at the individual level, to enable observation and prediction of the spread of the epidemic over time and space. Moreover, the effects of enforcing public health policies can be simulated using ABMs, allowing the adoption of 'what-if' tests to evaluate strategies for containment and mitigation.

While ABMs have significant potential for epidemic simulation, they are computationally demanding. Reconfigurable computing based on FPGAs supports fast processing. However, it is challenging to apply reconfigurable computing to the simulation of epidemics. In particular, the simulation algorithms for ABMs require intensive memory access over a large space. For example, to obtain the infection probability for an individual, distance-based simulation algorithms need to visit all other individuals in the worst case. This memory access pattern is challenging for FPGAs: their off-chip memory is large but slow, while their on-chip memory is fast but small. Even the latest FPGAs do not have on-chip memory sufficiently large to support simulation of agent-based epidemic models for the population of a small town.

This study addresses the above memory challenge for a common reconfigurable computing platform where a general-purpose computer collaborates with an FPGA-based accelerator. We propose a novel method to evaluate the infection probability using a limited amount of selected information. A key insight of the proposed method is to update the health status of an individual using the cumulative impact of a cluster of individuals. This makes it possible to update the health status by visiting a limited number of clusters rather than a large population of individuals. The proposed method therefore no longer requires the storage of the whole population. The result is that the on-chip memory usage no longer depends on the population size. As a result, an accelerator based on this approach can simulate a significantly larger population compared with conventional methods.

To the best of our knowledge, this paper presents the first reconfigurable accelerator for the simulation of large-scale epidemic spreading. The key contributions of the paper include the following:

- A novel way to evaluate the infection probability with constant on-chip memory usage. (Section III-A)
- Pre-computation and simulation algorithms for the host computer and the accelerator. (Section III-B)
- An efficient collaboration scheme between the accelerator and its host computer. (Section III-C)
- An empirical study on an FPGA-based platform on simulation speed and quality. (Section IV)

II. Background and related work

In this section, we review agent-based models for epidemic spreading and discuss existing acceleration solutions for general agent-based models.

A. Agent-based models for epidemic spreading

To understand the patterns and trends behind the spread of an epidemic, agent-based models provide a way to track the health status and interactions of people. There are two major ways to model the interactions between agents depending on whether or not the model considers the explicit geographical positions of agents and places [3].

Those models that do not involve the explicit geographical positions of locations typically use graph-based representations to describe the interaction pattern. The first graph-based method is called the labelled bipartite graph method [4] and employs a bipartite graph $G = (V_P, V_L, E)$ to represent the simulated world. The two sets of vertices V_P and V_L respectively represent the population and the set of locations. An edge in the edge set E connecting an individual and a location represents a visit of the individual to the location. Each edge carries a time interval as a label to mark the duration of the visit. A simulation algorithm can take advantage of the bipartite graph to compute the locations of agents at discrete time steps. Also, the simulation algorithm can compute the probability that healthy individuals get infected using the infection probability. The second graph-based method directly describes the relationship between individuals. In other words, these models treat the population as an undirected graph $G = (V, E)$ where V is the set of vertices standing for the population; E is the set of edges representing the contact between individuals in the population. Since the graph does not include temporal information, these models require additional methods to describe temporal changes of the agents. For instance, a widely-used approach to describe temporal changes is to incorporate the graph with a state-based infection model [5]–[7].

In contrast, in a spatially explicit model, the precise spatial locations of individuals and places provide an option to evaluate intervention options on the movement of agents [8]. In general, a short distance between two locations suggests a higher rate of movement [9]. In a spatially explicit model, the probability that an individual gets infected in a time step is

$$p_i = 1 - e^{-\lambda_i \Delta t} \tag{1}$$

where Δt is the length of a time step in the simulation; λ_i is the force of infection that individual i receives. For instance, the distance-based force of infection for individual i in [8], [10] is in the form:

$$\lambda_i = \frac{\sum_{k=0}^{N-1} h_k c_k \beta f(d(l_i, l_k))}{\sum_{k=0}^{N-1} f(d(l_i, l_k))} \tag{2}$$

where h_k is equal to 1 if individual k is infectious and is equal to 0 otherwise; β is the transmission rate in the general community; c_k is the severity of the symptom of individual k; l_i is the location of individual i; $d(x, y)$ is the geographical distance between x and y; $f(\cdot)$ is a function describing the relation between the intensity of transmission and the geographical distance. For instance, the function $f(\cdot)$ in [8] is a decreasing function in the following form:

$$f(d(l_i, l_k)) = \frac{1}{1 + (\frac{d(l_i, l_k)}{a})^b} \tag{3}$$

where a and b are positive real-valued parameters.

This paper focuses on a spatially explicit model where the probability of infection is based on unstructured contacts between individuals. Since the model is spatially explicit, we can extend the model with a mobility model [11] that considers the movement of people upon the availability of data. Also, although the model only covers unstructured contacts, we can use the techniques in [8] to couple the model with information on structured contacts.

B. Accelerated simulation for ABMs

Unfortunately, computational efficiency issues reduce the usability of agent-based models for large-scale epidemics. The analysis by Nakamura et al. [12] suggests that the time complexity of simulation can grow exponentially with the population size.

An existing approach to accelerate simulation algorithms is to design parallelised algorithms and run them on high-performance CPU and GPU clusters. Barrett et al. [13] propose a framework for CPU clusters to model the spread of epidemics for large social networks. The model representation takes advantage of disease-specific semantics to support automated decoupling of temporal and spatial dependence. The decoupling of dependence facilitates simulations across multiple computational nodes. The simulation algorithm can scale up to support large, realistic social networks with millions of vertices. Zou et al. [6] improve the network-based parallel epidemic simulation on GPU clusters from two aspects. The first aspect is to build efficient GPU implementations for graph algorithms. The second aspect is to improve the scalability by amortising the transmission latency over the total time of the computation.

Xiao et al. survey the agent-based models from the perspective of efficient computation [14]. The survey suggests that the acceleration of ABM simulation using reconfigurable computing platforms is challenging generally. In most existing reconfigurable computing solutions for ABMs, each agent occupies dedicated hardware resources. This approach, although straightforward, causes two problems. The first problem is the difficulty in inter-agent communication. Traditional bus communication usually follows the master-slave model. This model is typically inefficient for agent-based systems since the simulation in each time step involves a large number of transactions [15]. The second problem is the limited scalability with resources. The amount of logic resources required by simulation grows at least linearly with the population size. For instance, Gao et al. [16] present an accelerator for a graph-based epidemics model using FPGAs. Since each agent occupies dedicated logic resources, the experimental implementation on an Intel DE2i-150 FPGA Development Kit only supports at most 140 individuals.

This paper develops an FPGA-based simulator for large-scale agent-based epidemic models. Similar to existing hardware solutions for ABMs on clusters [13] and GPUs [6], our main objective is to accelerate the simulation procedure. However, compared with clusters and GPUs, the FPGA platform is considerably more power-efficient. Also, FPGA-based acceleration platforms become as accessible as GPUs in recent years since cloud service providers, including Amazon AWS, Intel DevCloud and Microsoft Azure, begin to offer FPGA-based instances. Compared with the only FPGA-based ABM simulator for epidemics we know [16] that simulates only 140 individuals, the proposed simulator can handle a large population of more than 10000 individuals.

III. Proposed method

This section presents the proposed agent-based model for epidemic spreading. Section III-A proposes a limited-memory method for health status update. The health status update method at the core idea in this study enables the simulation of a large population using only the on-chip memory on reconfigurable accelerators. Section III-B presents pre-computing and simulation algorithms for the host computer and the accelerator based on the method introduced in Section III-A. Section III-C explains a way to improve the collaboration efficiency between the host computer and the accelerator.

A. Limited-memory health status update

The simulation algorithm updates the health status of all agents in each time step in a probabilistic manner. This procedure is computationally expensive. Given a health and susceptible individual, the simulation algorithm evaluates its probability of being infected and changes its status to 'infected' with the calculated probability. In a spatially explicit model, the simulation algorithm needs to visit all susceptible individuals in the population to calculate the infection probability in the worst case. The calculation of infection probability for all susceptible individuals takes $O(N^2)$ time for each time step in the simulation.

Reconfigurable accelerators are less able to help accelerate the calculation of infection probabilities since there is a problem on the storage of the population. A reconfigurable computing platform usually has two types of memory. The on-chip memory offers high bandwidth but low capacity, while the off-chip memory provides high capacity but low bandwidth. In the calculation of an infection probability, the traverse through the whole population involves simple calculations and intensive serial memory access. Fortunately, it is possible to store the infectious population in the on-chip memory, so the memory bandwidth is unlikely to result in a speed bottleneck. However, the capacity of the on-chip memory would be too small to fit a reasonably-sized population for real-life scenarios. Alternatively, if the population is kept in the off-chip memory, then the storage capacity would be sufficient even if the population size is large. However, the low bandwidth would limit the speed of the traverse. To the best of our knowledge, there is no known method that addresses this problem.

This paper proposes a limited-memory health status update method which retrieves all required information from the on-chip memory during the computation. In particular, we store clustering information of the population rather than the population itself in the on-chip memory. Instead of computing the force of infection using Eq. 2, we approximate it using the clustering information with a set of parameters. A critical insight behind the idea is that the individuals in a real-world population often have a highly clustered spatial pattern. The distances between an individual and a cluster of other individuals tend to be similar. Since the force of infection in Eq. 2 is distance-based, we use the distance between the individual and the cluster centroid to approximate the cumulative contribution of all individuals in the cluster. In other words, the assumptions of spatially explicit models on epidemic spreading patterns remain unchanged. We approximate the infection probability to facilitate FPGA-based computation with an additional assumption that the population is highly clustered. As a result, it is only necessary to visit all cluster centroids to compute the force of infection instead of visiting all individuals in the population. Therefore, it is possible to update health status only using the on-chip memory as long as we can squeeze the cluster centroids and related auxiliary information in the on-chip memory.

To convert the idea of limited-memory health status update to algorithms, we propose to partition the population into J clusters and use the following approximate force of infection to replace the original one in Eq. 2:

$$\hat{\lambda}_i = \frac{\sum_{j=0}^{J-1} \upsilon_j g_j(d(l_i, \mu_j))}{\sum_{j=0}^{J-1} g_j(d(l_i, \mu_j))} \tag{4}$$

where μ_j is the centroid of cluster j and

$$g_j(d(l_i, \mu_j)) = \sum_{k=0}^{3} a_{j,k} d^{2k}(l_i, \mu_j) \tag{5}$$

Eq. 4 facilitates the proposed limited-memory health status update method. Specifically, Eq. 4 only visits J cluster centroids while Eq. 2 visits all the N individuals. In other words, the memory requirement of Eq. 4 scales with the number of clusters J rather than the population size N. We only need to select and maintain J centroids for the clusters and related parameters in the on-chip memory to compute the force of infection for any susceptible individual. We can choose a small value for J considering the capacity of the on-chip memory on the reconfigurable accelerator. The approximation accuracy reduces with a small J value. However, studies on clustering suggest that a small collection of clusters can be used to restore key information in the original data [17].

Eq. 5 evaluates the cumulative contribution of the force of infection using cluster centroids. The function takes a polynomial form which is different from its individual-based counterpart in Eq. 3. We design this particular polynomial function to balance the approximation accuracy and resource

usage. Although the polynomial function is structurally uncomplicated, its derivation includes non-trivial design considerations. The remainder of this section provides a sketch of the derivation.

A straightforward way to build the approximation for the force of infection is to copy the exact structure of $f(\cdot)$ from Eq. 3. This setting guarantees that the approximation error converges to zero when the number of clusters J reaches the population size N. However, we do not consider it wise to use the exact form of Eq. 3 because each centroid should have a different impact on susceptible agents depending on the spatial distribution of the infectious individuals in the cluster. Therefore, instead of using an identical force function $f(\cdot)$ for all infected individuals, it is necessary to assign a different function $g_j(\cdot)$ to each cluster. We choose to use a polynomial function $g_j(\cdot)$ in Eq. 5 for each cluster to approximate the cumulative contribution of the force of infection. The polynomial function avoids expensive arithmetic operations such as divisions. On the other hand, although $d(l_i, \mu_j)$ can be derived from $d^2(l_i, \mu_j)$ with a square root operation, performing the square root operation on an FPGA is costly and inaccurate. The form of $g_j(\cdot)$ in Eq. 5 can avoid such a square root operation.

To balance the considerations on mathematics and resources, we choose the lowest order that can preserve the original concavity. As $g_j(\cdot)$ works with squared distance space, we focus on the function $\phi(s) = f(\sqrt{s})$ which has two intervals with different concavity. Therefore, we use the cubic polynomial of $d^2(l_i, \mu_j)$ to construct $g_j(\cdot)$.

B. Pre-computation and accelerated simulation

The host computer needs to pre-compute model parameters before the simulation. The parameters include the centroids of the clusters, the coefficients $a_{0..(J-1),0..3}$ for $g_j(\cdot)$ and the dynamic parameters $\upsilon_{0..(J-1)}$ to compute the force of infection. During the simulation, the host computer maintains the population while the accelerator on the FPGA platform updates the health status for the population using the model parameters.

The search for the globally optimal parameters is an NP-hard problem. In this study, we propose a greedy method to compute the parameters from $f(\cdot)$ in two phases. The first phase is clustering. The host computer identifies J clusters from the population. Given the population and the number of clusters J, this clustering procedure is invoked only once. The clustering metric is independent of the parameters $a_{0..(J-1),0..3}$ or $\upsilon_{0..(J-1)}$. The second step is parameter estimation, which takes the population and the centroids of clusters to compute $a_{0..(J-1),0..3}$ and $\upsilon_{0..(J-1)}$. In other words, the parameter estimation procedure considers the clustering result from the first phase as optimal and does not revise it.

Fig. 1 shows an example of the two-step pre-computation workflow. Given the population in Fig. 1(a), the pre-computation algorithm identifies 5 clusters, as shown in Fig. 1(b). The points in the same colour represent the individuals in the same cluster. A cross in the figure standard for the centroid of the cluster it lays on. Fig. 1(c) demonstrates a parameter estimation result where each circle marks the boundary where the corresponding $g_j(x)$ decreases by half.

Algorithm 1: Pre-computation

Input: I: set of individuals, J: number of clusters
Output: Parameters $\mu_{0..(J-1)}$, $a_{0..(J-1),0..3}$, $\upsilon_{0..J}$

$\theta_{0..(J-1)} \leftarrow \text{IdentifyClusters}(I, J)$
$\mu_{0..(J-1)} \leftarrow \text{ExtractCentroids}(\theta)$
for $j \in 0..(J-1)$ **do**
  $a_{j,0..3} \leftarrow \arg\min_{a_{j,0..3}}(E_a)$
  $\upsilon_j \leftarrow \arg\min_{\upsilon_j}(E_\upsilon)$
return $\mu_{0..(J-1)}$, $a_{0..(J-1),0..3}$, $\upsilon_{0..J}$

Algorithm 1 shows detailed steps for the two-phase computation. The two lines before the for-loop correspond to the clustering phase. The clustering method used in this paper is K-means [18]. The for-loop itself corresponds to the parameter estimation phase. The two optimisation objectives in this phase are:

$$E_a = \sum_{i=0}^{J-1} \left(g_j(d(\mu_i, \mu_j)) - \sum_{k=0}^{N-1} \iota_{j,k} f(d(\mu_i, l_k)) \right)^2 \tag{6}$$

$$E_\upsilon = \sum_{i=0}^{J-1} \left(\upsilon_j g_j(d(\mu_i, \mu_j)) - \sum_{k=0}^{N-1} \iota_{j,k} h_k c_k \beta f(d(\mu_i, l_k))) \right)^2 \tag{7}$$

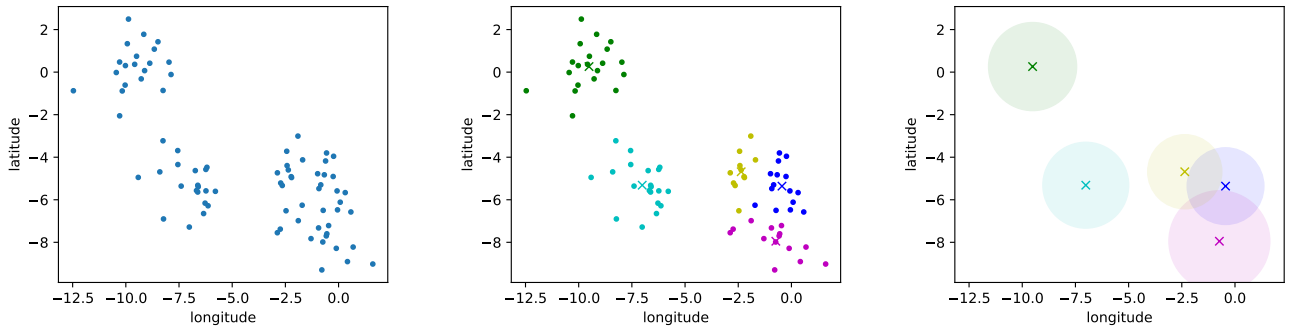
where $\iota_{j,k}$ is equal to 1 if individual k is in cluster j and is equal to 0 otherwise. The minimisation problems for E_a and E_υ are standard least-square problems with closed-form solutions.

Algorithm 2: Health status update (on FPGA)

Input: h_j: health status, Δt: time step length
Output: $h^j_{0..(N-1)}$: updated health status

$s_\top \leftarrow 0$
$s_\perp \leftarrow 0$
for $j \in 0..(J-1)$ **do**
  $w \leftarrow g_j(d(l_i, \mu_j))$
  $s_\top \leftarrow u + w\upsilon_j$
  $s_\perp \leftarrow u + w$
$\hat{\lambda} \leftarrow \frac{s_\top}{s_\perp}$
$h_j^+ \leftarrow \begin{cases} 0 & \text{with probability } e^{-\hat{\lambda}\Delta t} \\ 1 & \text{otherwise} \end{cases}$

Algorithm 2 shows the health status update procedure for one individual. The loop implements the evaluation of $\hat{\lambda}_i$ in Eq. 4. The two variables $s_\top$ and $s_\perp$ are respectively the accumulators for the numerator and the denominator on the right-hand-side of Eq. 4. The calculations after the loop correspond to the sampling process of h_k following Eq. 1. Since all iterations of the loop are independent, we unroll



(a) Population (b) Clustered population (c) Parameter estimation

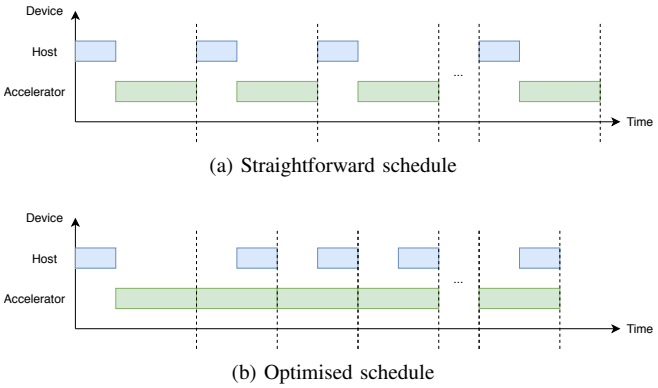
Fig. 1: Pre-computation on host computer

the loop so that the accelerator runs U iterations in parallel. Therefore, a larger value for U generally results in a higher simulation throughput. As a result, the resource-intensive part in Algorithm 2 is the loop even if the computation in a single iteration only involves the evaluation of a polynomial. The operations after the loop use fewer resources than the loop itself. It is only necessary to deploy one hardware block for these operations because this block only needs to work once after J iterations of the loop.

Given a single FPGA chip, we recommend implementing a single copy of Algorithm 2 and maximise U to improve the throughput. Given multiple FPGA chips, we can implement one copy of Algorithm 2 on each chip to take advantage of the data parallelism, as long as each chip can keep a sufficient number of clusters in its on-chip memory to maintain enough approximation accuracy. Since there are no sequential dependencies between the individuals in the population, the host computer can split the population and stream the sub-populations to different FPGA chips. During the simulation, the data exchanged between the host computer and the FPGAs only include the locations of individuals, the health status and the parameter vector $\upsilon_{0..(J-1)}$. The FPGAs do not need to communicate with each other. Therefore, we can easily take advantage of data parallelism when more units of reconfigurable hardware are available. Also, if we assume that (a) the host computer can distribute the model parameters and collect the health status for all processing units in parallel and (b) the population is sufficiently large, then the simulation throughput should scale linearly with the number of processing units. With multiple FPGAs, we may scale up the simulation of an epidemic at country level within limited time. An illustration of this linear scaling is presented in Section IV-C.

C. Efficient host-accelerator collaboration

At the beginning of each time step, we need to make sure that the parameter vector $\upsilon_{0..(J-1)}$ in the on-chip memory of the accelerator is up-to-date. A straightforward way is to take the health status at time step $(t-1)$ to update $\upsilon_{0..(J-1)}$ for time step t. This straightforward workflow is based on the assumption that an individual i can infect others as soon as individual i itself gets infected. The assumption can simplify the simulation logic, but the data dependence negatively affects the collaborating efficiency. Fig. 2(a) is a Gantt chart for this collaboration plan. The data transmission time is not shown for simplicity. The host and the accelerator take turns to perform computation. The host computer starts to compute $\upsilon_{0..(J-1)}$ after the accelerator returns the health status of the previous time step. The accelerator needs to wait for $\upsilon_{0..(J-1)}$ before updating the health status. We consider the collaboration in Fig. 2(a) to be inefficient as the host and the accelerator can never run in parallel.



(a) Straightforward schedule

(b) Optimised schedule

Fig. 2: Host-accelerator collaboration

The key problem that hampers parallelisation is the assumption that an individual can become infectious immediately after getting infected. However, we notice that the assumption does not hold biologically. Instead, there is a gap between the time that an individual get infected and the time that the newly infected individual is ready to infect others. This biological conclusion turns out to be critical in the optimisation of the workflow. Critically, it is possible to optimise the host-accelerator collaboration schedule if the gap time is larger than one time step in the simulation. We assume that an individual infected at time step t is unable to infect others at time step

$(t+1)$. Instead, let $q \in (1, \infty)$ be an integer such that an individual infected in time step t starts to infect others at the beginning of time step $(t+q)$.

Fig. 2(b) shows an optimised schedule for this revised collaboration. The schedule works as follows. At time step 0, the host computer starts by computing the initial $v_{0..(J-1)}$. The accelerator updates the health status of all individuals using the initial $v_{0..(J-1)}$. At time step $t \in (0, q-1]$, the accelerator can still use the initial $v_{0..(J-1)}$ to update the health status of the individuals. While the accelerator is working, the host can compute $v_{0..(J-1)}$ for the current time step t. At time step $t \in [q, +\infty)$, $v_{0..(J-1)}$ for time step $(t-q)$ must be available. The accelerator can use the this version of $v_{0..(J-1)}$ to update the health status. Similar to the previous case, the host can compute $v_{0,(J-1)}$ for time step t while the accelerator is busy.

IV. Evaluation

In this section, we present an empirical evaluation of the proposed model. We first investigate the accuracy–efficiency trade-off of the limited-memory health status update method. Then, we show the simulation results for a synthetic epidemic on statistical reliability and computational efficiency.

A. Experimental setup

We implement the health status update in Algorihtm 2 using OpenCL 1.2, compile it with the Intel FPGA OpenCL SDK toolchain 19.4 and map the kernel on an Intel Arria 10 GX FPGA development platform. The platform contains an Arria 10 GX 10AX115S2F45I1SG FPGA. The maximum loop unroll factor we achieve is $U = 120$. The corresponding resource usage is shown in Table I. The workstation hosting the FPGA platform has a six-core Intel Core i5-9400F CPU running at 2.90GHz and 32GB DDR-3333 memory. A PCIe x8 edge connector links the FPGA platform to the host. We use 32-bit floating point numbers across the design. The software reference is an implementation of the model in [8]. As we focus on spatially explicit models, we disable the non-spatial force of infection in the source code before compilation. The software reference is written in the C programming language and compiled with the Intel C compiler 19.1 with the '-O2' optimisation flag.

TABLE I: Resource usage

Resource	ALMs	FFs	RAMs	DSPs
Usage	572410	831570	1930	1211
Usage(%)	67%	49%	71%	80%
Total	854400	1708800	2713	1518

B. Accuracy versus computational burden

The key insight of the proposed method is that a limited number of clusters are sufficient to reconstruct the force of infection. We analyse the trade-off between the number of clusters J and the approximation error. The number of clusters J directly determines the computational burden at each time step. With a fixed U, the execution time for each time step increases linearly with J. The approximation error affects how much the approximated version follows the same behaviour as the original version.

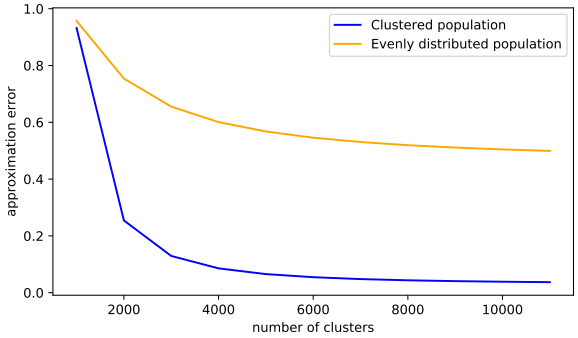
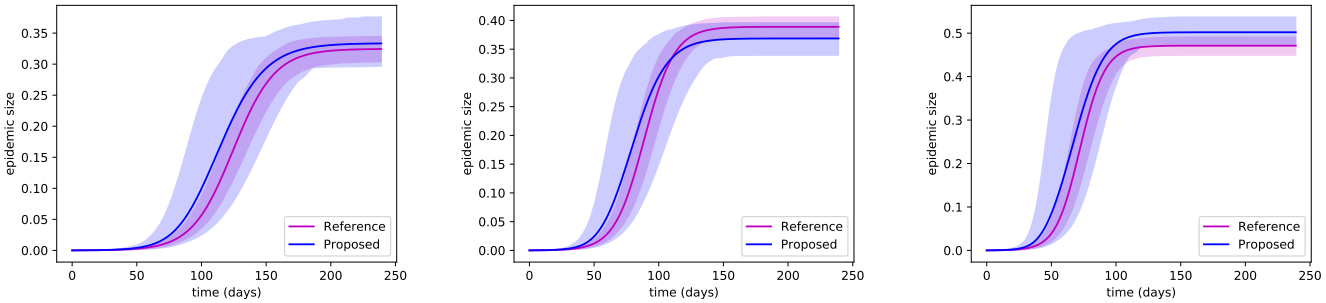


Fig. 3: Approximation error versus number of clusters

An ideal way to observe the impact of J on the model quality is to observe the model quality with different values for J. However, to the best of our knowledge, there is no generally accepted way to quantify the quality of ABM simulation for epidemics. As a result, we study the impact of J indirectly by looking at how the approximation error of Eq. 4 varies with the number of clusters. The locations of N=100000 individuals are generated with the method in [19]. The result is shown in the blue curve in Fig. 3. The error converges quickly towards zero, while suggests that a small number of clusters can provide sufficient accuracy for a large population. However, we believe that the convergence depends on the general assumption that individuals in the population form clusters. If the individuals distribute evenly in space, the error would converge more slowly. To confirm this, we plot a curve for an evenly distributed population in orange in Fig. 3. The curve shows that we need to use a larger number of clusters for an evenly distributed population to achieve the same level of accuracy with a highly clustered population. However, in a realistic simulation, the slow convergence of the error is unlikely to appear since individuals in real-life populations do tend to form clusters [20].

C. Accelerated simulation

We use the synthetic epidemic settings [8] and the population in the previous experiment to evaluate the quality and speed of the experimental implementation. We use $J = 10000$ in all experiments. Since there is no universally accepted method to judge the quality of ABM models for epidemics, we directly put the results from both models side by side following [8]. Fig. 4 shows how the epidemic size changes with time in the simulation for both models. Table II presents the average epidemic sizes and standard deviations every 30 days. The three sub-figures in Fig. 4 correspond to different basic reproductive ratios $R_0 = 1.5$, 1.9 and 2.3. Each growth curve is the average epidemic size based on 200 repetitions. The band around the curve marks the 95% confidence interval. In all the simulation settings, the growth trends of the epidemic size appear as S-shaped curves. Regarding the averaged trend,



(a) $R_0 = 1.5$ (b) $R_0 = 1.9$ (c) $R_0 = 2.3$

Fig. 4: Epidemic size versus spreading time

TABLE II: Numeric results on epidemic size

Day	Reference $R_0 = 1.5$	Proposed $R_0 = 1.5$	Reference $R_0 = 1.9$	Proposed $R_0 = 1.9$	Reference $R_0 = 2.3$	Proposed $R_0 = 2.3$
30	0.0006 ± 0.0002	0.0008 ± 0.0003	0.0018 ± 0.0005	0.0027 ± 0.0014	0.0039 ± 0.0011	0.0071 ± 0.0062
60	0.0048 ± 0.0012	0.0078 ± 0.0041	0.0261 ± 0.0071	0.0538 ± 0.0454	0.0921 ± 0.0296	0.1638 ± 0.1195
90	0.0308 ± 0.0074	0.0574 ± 0.0369	0.1953 ± 0.0362	0.2458 ± 0.0774	0.4008 ± 0.0338	0.4353 ± 0.0663
120	0.1340 ± 0.0252	0.1893 ± 0.0683	0.3615 ± 0.0155	0.3506 ± 0.0292	0.4677 ± 0.0113	0.4974 ± 0.0176
150	0.2658 ± 0.0219	0.2915 ± 0.0436	0.3868 ± 0.0098	0.3670 ± 0.0162	0.4711 ± 0.0111	0.5019 ± 0.0169
180	0.3138 ± 0.0127	0.3245 ± 0.0217	0.3888 ± 0.0097	0.3685 ± 0.0156	0.4712 ± 0.0111	0.5020 ± 0.0170
210	0.3229 ± 0.0117	0.3319 ± 0.0188	0.3889 ± 0.0097	0.3686 ± 0.0155	0.4712 ± 0.0111	0.5021 ± 0.0170
240	0.3244 ± 0.0117	0.3334 ± 0.0197	0.3889 ± 0.0097	0.3686 ± 0.0155	0.4712 ± 0.0111	0.5021 ± 0.0170

the proposed model makes little difference from the reference model. However, the main difference between the two models is the confidence interval. The proposed model tends to have wider bands than the reference model, which means that the epidemic size in the proposed model spreads wider across multiple simulations. Therefore, the trend of the epidemic size from the proposed model may be less stable with a small number of repetitions. However, the observation is unlikely to cause problems as a practical simulation procedure of ABMs involves 100-1000 repetitions [13].

TABLE III: Execution time and speedup

Loop unroll factor U	Exec. time (seconds)	Speed-up (one core)	Speed-up (six cores)
10	21.98	5.1	1.1
20	10.99	10.3	2.3
30	7.33	15.4	3.4
40	5.49	20.6	4.6
50	4.40	25.7	5.7
60	3.66	30.8	6.9
70	3.14	36.0	8.0
80	2.75	41.1	9.1
90	2.44	46.3	10.3
100	2.20	51.3	11.4
110	2.00	56.5	12.6
120	1.83	61.7	13.7

We measure the simulation throughput by the number of simulated time steps per second. On average, the simulation of the reference model takes respectively 112.91 and 25.10 seconds for each time step on one CPU core and six CPU cores. The execution time with the accelerator is shown in Table III. In this table, U is the loop unroll factor discussed in Section III-B; the execution time is in seconds. In all experiments, the optimised collaboration scheme amortises the pre-computation time on the host computer over the total execution time. The execution time for one time step drops from around 21.98 seconds to 1.83 seconds as U increases from 10 to 120. The maximum speedup compared with the software reference is around 62 times over one CPU core and 14 times over six CPU cores.

TABLE IV: Number of Arria 10 FPGAs to simulate a 180-day epidemic within 24 hours

Region	Population	# FPGAs
USA	328 million	9
EU	446 million	12
UK	67 million	2
China	1.39 billion	43
Japan	127 million	5

This paper focuses on the acceleration with a single FPGA chip. However, we can scale up to support country-level simulations using multiple FPGAs. We estimate the number of Intel Arria 10 FPGAs to simulate a 180-day epidemic within 24 hours in Table IV with the following assumptions: (a) a one-day time step can provide sufficient simulation accuracy [8]; (b) each FPGA has enough on-chip memory space to support accurate approximation; (c) the host computer can update the parameter vector $\eta_{0..(J-1)}$ for all FPGAs in parallel; (d) the optimised collaboration scheme can still amortise the time for parameter update time over the total execution time.

V. Conclusions and future work

In the simulation of spatially explicit agent-based epidemic models, the dependence between the population size and the on-chip memory usage results in a difficult trade-off. If the on-chip memory stores the population, then the population size would be too small to simulate populations in the real world. However, if we allow some agents to stay off-chip at any time, then the low bandwidth would limit the simulation throughput. We avoid the difficult trade-off by deriving a novel method to update the health status of the population during the simulation. A unique feature of our new approach is that the on-chip memory requirement no longer depends on the size of the simulated population. This feature significantly enlarges the population that a reconfigurable computing platform can simulate efficiently. We also propose algorithms for pre-computation and parallelised health status update. In addition to the computation of the infection probability, we take advantage of the latency period of disease infection to develop an efficient collaboration scheme between the accelerator and its host computer.

There are at least three directions of future work. The first direction is to optimise the hardware design. This paper focuses on the algorithmic aspects of acceleration. There is still a lot to do with performance improvement on implementation-level optimisations. For instance, the experimental implementation uses floating-point numbers. However, the algorithmic optimisations proposed in Section III-B, including the avoidance of square roots and exponential functions, would result in more significant improvement with fixed-point numbers than floating-point numbers. The second direction is to calibrate the model with data and perform large-scale simulations under practical settings. Although the main statistical objective in this paper is to predict the trend accurately, the design of a practical simulation tool should involve the capability of 'what-if' tests and interventions, as well as a good trade-off between speed and accuracy. The third and perhaps most interesting and useful direction is to generalise the model and the acceleration strategy to support other types of non-epidemic ABMs [21]. This would appear to offer widespread social benefit beyond the immediate and critical importance of the global impact of the COVID-19 pandemic on health. Our approach could provide valuable improvement in the speed of evaluation of the impact of pandemics on the economies, behaviours and social structures of the entire world, especially in areas such as how other social and economic structures can spread ideas, information and beliefs even more rapidly than a virus can spread.

Acknowledgements

The support of UK EPSRC (grant number EP/L016796/1, EP/I012036/1, EP/L00058X/1, EP/N031768/1 and EP/K034448/1) and Intel is gratefully acknowledged.

References

[1] J. M. Last, S. S. Harris, M. C. Thuriaux, and R. A. Spasoff, *A dictionary of epidemiology*. International Epidemiological Association, Inc., 2001.

[2] K. J. Rothman, S. Greenland, and T. L. Lash, *Modern epidemiology*. Lippincott Williams & Wilkins, 2008.

[3] N. M. Ferguson, D. A. Cummings, S. Cauchemez, C. Fraser, S. Riley, A. Meeyai, S. Iamsirithaworn, and D. S. Burke, "Strategies for containing an emerging influenza pandemic in southeast asia," *Nature*, vol. 437, no. 7056, pp. 209–214, 2005.

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

[5] F. S. Tabataba, B. Lewis, M. Hosseinipour, F. S. Tabataba, S. Venkatramanan, J. Chen, D. Higdon, and M. Marathe, "Epidemic forecasting framework combining agent-based models and smart beam particle filtering," in *2017 IEEE International Conference on Data Mining*. IEEE, 2017, pp. 1099–1104.

[6] P. Zou, Y.-s. Lü, L.-D. Wu, L.-L. Chen, and Y.-P. Yao, "Epidemic simulation of a large-scale social contact network on GPU clusters," *Simulation*, vol. 89, no. 10, pp. 1154–1172, 2013.

[7] K. R. Bisset, J. Chen, S. Deodhar, X. Feng, Y. Ma, and M. V. Marathe, "Indemics: An interactive high-performance computing framework for data-intensive epidemic modeling," *ACM Transactions on Modeling and Computer Simulation*, vol. 24, no. 1, pp. 1–32, 2014.

[8] M. Ajelli, B. Gonçalves, D. Balcan, V. Colizza, H. Hu, J. J. Ramasco, S. Merler, and A. Vespignani, "Comparing large-scale computational approaches to epidemic modeling: agent-based versus structured metapopulation models," *BMC infectious diseases*, vol. 10, no. 1, p. 190, 2010.

[9] C. Viboud, O. N. Bjørnstad, D. L. Smith, L. Simonsen, M. A. Miller, and B. T. Grenfell, "Synchrony, waves, and spatial hierarchies in the spread of influenza," *science*, vol. 312, no. 5772, pp. 447–451, 2006.

[10] M. Ajelli and S. Merler, "The impact of the unstructured contacts component in influenza pandemic modeling," *PLoS One*, vol. 3, no. 1, 2008.

[11] E. Frias-Martinez, G. Williamson, and V. Frias-Martinez, "An agent-based model of epidemic spread using human mobility and social network information," in *2011 IEEE third international conference on privacy, security, risk and trust and 2011 IEEE third international conference on social computing*. IEEE, 2011, pp. 57–64.

[12] G. M. Nakamura, A. C. P. Monteiro, G. C. Cardoso, and A. S. Martinez, "Efficient method for comprehensive computation of agent-level epidemic dissemination in networks," *Scientific reports*, vol. 7, no. 1, pp. 1–12, 2017.

[13] C. L. Barrett, K. R. Bisset, S. G. Eubank, X. Feng, and M. V. Marathe, "Episimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks," in *SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. IEEE, 2008, pp. 1–12.

[14] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "A survey on agent-based simulation using hardware accelerators," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–35, 2019.

[15] E. A. Gerlein, T. M. Mcginnity, A. Belatreche, and S. Coleman, "Network on chip architecture for multi-agent systems in FPGA," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 10, no. 4, pp. 1–22, 2017.

[16] T. Gao, "FPGA of acceleration of stochastic simulation," Master's thesis, Cornell University, 2014.

[17] A. Kaarna, P. Zemcik, H. Kalviainen, and J. Parkkinen, "Compression of multispectral remote sensing images using clustering and spectral reduction," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 38, no. 2, pp. 1073–1082, 2000.

[18] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[19] T. Smieszek, M. Balmer, J. Hattendorf, K. W. Axhausen, J. Zinsstag, and R. W. Scholz, "Reconstructing the 2003/2004 H3N2 influenza epidemic in Switzerland with a spatially explicit, individual-based model," *BMC infectious diseases*, vol. 11, no. 1, p. 115, 2011.

[20] S. Riley, "Large-scale spatial-transmission models of infectious disease," *Science*, vol. 316, no. 5829, pp. 1298–1301, 2007.

[21] L. Cui, J. Chen, Y. Hu, J. Xiong, Z. Feng, and L. He, "Acceleration of multi-agent simulation on FPGAs," in *International Conference on Field Programmable Logic and Applications*. IEEE, 2011, pp. 470–473.