2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS) | 978-1-6654-1913-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/AICAS51828.2021.9458570

# FPGA-accelerated Agent-Based Simulation for COVID-19

Lei Fu, Ce Guo and Wayne Luk

Imperial College London, United Kingdom
Email: {lei.fu18, c.guo, w.luk}@imperial.ac.uk

***Abstract*—Agent-based models (ABMs) can provide realistic dynamics for epidemics at the individual level so that users can observe and predict the spreading pattern and the effectiveness of intervention over time and space. This paper proposes an FPGA-based accelerator for agent-based epidemic modeling for COVID-19. The optimizations enabling the effective acceleration of the simulation procedure are presented. The key idea is to partition the calculation properly to decouple the on-chip resource usage from the population size. Also, an algorithmic adaptation is proposed to reduce the latency caused by conditional branches within loops. An experimental implementation on an Intel Arria 10 GX 10AX115S2F45I1SG FPGA running at 240MHz achieves 2.2 and 1.9 times speed-up respectively over a CPU reference using 10 cores on an Intel Xeon Gold 6230 CPU and a GPU reference on an Nvidia GeForce RTX 2080 Ti GPU.**

## I. Introduction

COVID-19 is a highly contagious epidemic that transmits mainly via respiratory droplets. Decision support systems that aid people in predicting and intervening in the spread of the epidemic are in great need. Among modern artificial intelligence techniques, a useful approach to build such decision support systems is agent-based modeling.

Agent-based modeling is widely used as an AI approach to analyze stochastic and chaotic systems. Over the last decade, agent-based models for epidemics have become popular due to their ability to incorporate individual-level dynamics and complex interventions. A critical calculation for agent-based models is the simulation. Given the agents' setup and their actions in each time step, the simulation procedure updates the states of the agents through time. The simulation algorithm keeps track of each agent and for further analysis.

This paper introduces an FPGA design to accelerate the simulation of an agent-based model for COVID-19. Specifically, the design efficiently calculates a key quantity in the simulation process, namely the household force of infection (FOI) in the CovidSim model [1]–[3]. In the proposed approach, the on-chip resources usage is independent of the maximum size of the simulated population. As a result, the approach facilitates the simulation of large populations. The main contributions of the paper include:

- A cache mechanism decoupling the on-chip resource usage from the population size.
- An algorithmic adaptation removing conditioning statements from the simulation procedure.
- An experimental study comparing an FPGA implementation against a multi-core CPU and a GPU.

## II. Background

Computational methods to model the spread of epidemics include deterministic models [4], structured metapopulation models [5] and agent-based models [3]. Agent-based models are unique because they can keep track of individuals' actions, behaviors, and status through time. The individual-level modeling allows the users to encode information such as the geological distribution of the population. Besides, agent-based models allow the users to simulate the scenarios with human intervention to analyze containment and mitigation effectiveness.

Accelerated simulation of agent-based models on FPGAs is challenging in general [6]. Although there have been a few solutions to accelerate agent-based epidemic models using CPU and GPU clusters [7], [8], there are only two known studies on the simulation of epidemics using FPGAs. Both of them have serious limitations.

The first study is on a design of the Susceptible-Exposed-Infectious-Removed (SEIR) model [9]. In this approach, each individual in the population occupies dedicated on-chip resources. An advantage of this approach is that the simulation of all agents can take place in parallel. However, the on-chip resource usage grows with the population size. In other words, the maximum population size depends on the availability of on-chip resources. For instance, the experimental implementation in [9] on an Intel Cyclone IV 4CX150 FPGA can only simulate 140 agents. Practically, an agent-based model usually has a minimum population size to generate a proper collective behavior. When the population requires more resources than the device can offer, it is impossible to simulate the model. Our proposed design avoids the problem by decoupling the correlation between the resource usage and the population size.

The second study is a large-scale FPGA-based simulator based on a space-explicit model [10]. The model comes from [5] with hardware-oriented adaptations. This design uses clustering techniques to compute an approximate infection probability for each individual at each time step. This approximation method allows the information required by the FOI to fit in limited on-chip memory. As a result, the design can support a far larger population than the one in [9]. However, the clustering-based approximation reduces the reliability of simulation so that the variance of the infected population over multiple runs is significantly larger than that of the original model in [5]. Moreover, since the model in [10] is space-explicit, it requires a geological location model for the agents. Therefore, the model is difficult to calibrate due to the randomness of human behavior and the lack of data. Our proposed design avoids these problems by adopting a well-studied epidemic model with public-domain software.

## III. Design and implementation

### A. Reference model

The CovidSim [1]–[3] micro-simulation model developed by MRC centre for Global Infectious Disease Analysis hosted at Imperial College London is used as software reference. CovidSim models the transmission dynamics and severity of COVID-19 throughout a spatially and socially structured population over time, which also takes intervention policies and healthcare into account.

The propagation of newly infected cases has been spotted as the performance bottleneck. This process takes around 85% of total execution time and has been called around 20,000 times for a country with around 10M population. After detailed analysis on its memory access pattern, it has been found that the process of selecting infectors mostly follows a sequential pattern and there is little serial data dependency during propagation. These features make it a suitable model reference.

Algorithm 1 shows the abstracted simulation procedure of each propagation. It can be divided into two components. The first component calculates the probability of infection for each individual and decide whom to infect, while the second component processes through the infection queue generated and update each individual's status. The main indicator is called FOI (force of infection). It is expressed as the product of two factors: infectiousness and susceptibility. Infectiousness takes into account their age, place, vaccination status, etc. Susceptibility indicates a person's susceptibility to another person.

Although the design objective is to accelerate the household FOI for COVID-19, it is possible to use the design in other fields. For instance, the design requires little modification to accelerate epidemic simulation with household infections such as the flu. Also, the essential assumption for the design is that people with an infectious person in the same household can get infected. The assumption works in general for places where people stay together for a significant time, like workplaces and schools. Therefore, we may extend the proposed approach to place-related infections in general.

### B. Kernel design and implementation

Parameters fed into the kernel include global parameters for the simulation, records of all individuals and households, information about institutions that people may belong to and quarantine status of people.

Algorithm 2 presents the calculations in the kernel. It first selects the households with more than one non-traveling individual. Then it calculates the infectiousness of the selected household and scales up the infectiousness of that household if individuals are absent from their places. After that, it iterates through all individuals in the household and calculates the susceptibility and FOI (force of infection) for those who are not infected.

**Algorithm 1:** Abstracted Simulation Procedure

```
1  foreach infected household $h_i \in Households$ do
2      $inf_i \leftarrow$ CalculateInfectiousness($h_i$);
3      if any of the individuals in the selected household
         are absent from places then
4          $inf_i \leftarrow inf_i * PlaceCloseHouseholdRelContactConstant$;
5      foreach individual $p_j \in h_i$ do
6          if $p_j$ is uninfected and not travelling then
7              $sus_j \leftarrow$ CalculateSusceptibility($h_i, inf_i$);
8              $FOI_j \leftarrow inf_i * sus_j$;
9              if $FOI_j >$ a random float between 0 and 1
                 then
10                 Calculate the generation of infection;
11                 Push $p_j$ to infection queue;
```

### C. Chunk processing and cache system

One major challenge for processing large usage is the positive correlation between the usage of on-chip resource and the population size. Although records for persons and households are parsed and stored continuously on global memory, the simulation procedure often includes frequent irregular memory access when indexing between them, which could dramatically increase usage of memory blocks and latency of execution. Fig. 1 shows the memory access pattern between households and individuals.

To decouple the correlation, one reasonable approach is to divide parameter arrays into independent chunks. In this way, each chunk could be cached and processed separately to minimize communications between on-chip and off-chip memory. Because of the clustering nature of persons and households, related statuses are often stored in continuous spaces and parameter arrays for individuals are often aligned accordingly with the indexing of their households.

However, the unpredictable pattern of starting index and size of each cluster makes partitioning infeasible. If individuals are partitioned into blocks with fixed size, there could be a magnitude of cache misses, which would cause additional communications between on-chip and off-chip memory.

A preprocessing routine is therefore proposed to reorganize parameter arrays. It iterates through all parameters array while keeping track of the indexing between them. A padding is added to each cluster, leveraging it to a fixed size. Irregular indexing would also be detected and mitigated during the

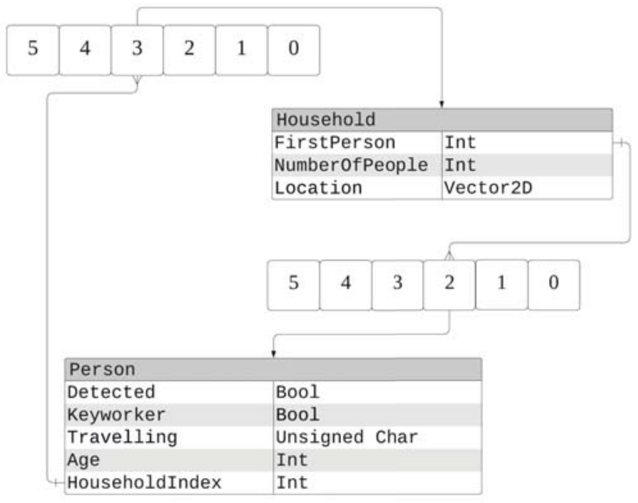

Fig. 1: Memory access pattern between household and person

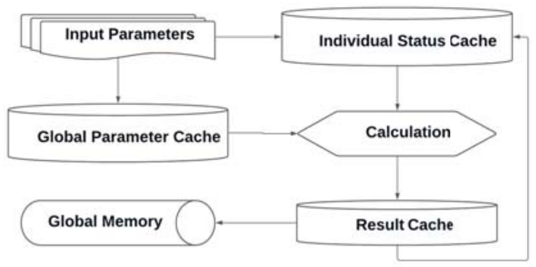

Fig. 2: Cache system around the calculation component

process. The routine takes $\mathrm{O}(N)$ time and $\mathrm{O}(N)$ additional space with a small constant. It would be activated only once for each simulation procedure. The time complexity of the simulation is at least $\mathrm{O}(NT)$ where $T$ is the number of time steps. Therefore, the additional $\mathrm{O}(N)$ time for preprocessing does not change the overall time complexity of the simulation procedure. On the other hand, the additional space for padding only increases the off-chip memory usage, while the on-chip memory usage keeps unchanged.

After regularizing all parameter arrays with individual statuses, a cache system is implemented with three categories: global parameter cache, individual status cache and result cache, as shown in Fig. 2. Fixed size parameters, including the Age Group Susceptibility and WAIFW (who acquires infection from whom) matrix, are stored in global parameter cache. Before processing each chunk, relevant parameters are first parsed and stored in individual status cache. In this way, all calculation components are separated from off-chip memory. The result cache is activated when all possible calculations are finished for an individual and would be dumped to off-chip memory after a constant number of chunks are processed.

### D. Control flow conversion

Another challenge that limits the performance is the latency caused by control flows. As the simulation process includes frequent status checks, there are lots of nested conditional branches within loops. Those branches cause difficulties for static optimization and exhaust on-chip logical units.

An algorithmic adaptation is therefore proposed to mitigate this effect and reduce latency. Instead of justifying whether an individual should be infected, the target for each simulation procedure is directed to calculate the probability for one individual to be infected. FOI is used as the basis for the final probability. Most conditional branches within loops are thus converted to arithmetic operations that measure their impact on the resulting probability.

**Algorithm 2:** Abstracted Kernel Execution Procedure

```
1  Store Age Susceptibility and WAIFW matrix in global
     parameter cache;
2  foreach chunk of households $c_i \in Households$ do
3      Store related individual records in individual status
         cache;
4      foreach infected household $h_i \in c_i$ do
5          $inf_i \leftarrow$ CalculateInfectiousness($h_i$);
6          foreach individual $p_j \in h_i$ do
7              $inf_i *=$ (1 + a boolean indicating absence);
8          foreach individual $p_j \in h_i$ do
9              $sus_j \leftarrow$
                 CalculateSusceptibility($h_i, inf_i$);
10             $FOI_j \leftarrow inf_i * sus_j$;
11             Calculate the generation of infection $g_j$;
12             $r_j \leftarrow FOI_j$ * other related individual
                 records of $p_j$;
13             Store $r_j$ and $g_j$ in result cache;
14     Dump result cache to off-chip memory;
```

## IV. Evaluation

This section presents an evaluation of an experimental implementation. We evaluate the accelerator using an FPGA-based development platform and compare its speed against CPU and GPU implementations.

### A. Experiment setup

We compile and run the hardware design using the FPGA acceleration platform on the Intel DevCloud. The host has an Intel Xeon Gold 6230 CPU running at 2.10GHz. The hardware kernel is developed in OpenCL 1.2. We compile the kernel using the Intel FPGA SDK for OpenCL 19.4. The CPU communicates with an Intel Programmable Acceleration Card (PAC) based on an Arria 10GX 10AX115S2F45I1SG FPGA built on 20nm technology. In all experiments, we clock the FPGA at 240MHz. The resource usage of the implementation is shown in Table I. Since the on-chip resource usage is independent of the population size, the resource usage stays unchanged when the population size grows.

We also evaluate and compare the execution time of one simulation procedure on the CPU and GPU platforms. The

TABLE I: On-Chip Resource Usage

| Resource | Available | Used | Used (%) |
|---|---|---|---|
| ALUTs | 854400 | 520166 | 61% |
| RAMs | 2713 | 1145 | 41% |
| FFs | 1708800 | 300324 | 30% |
| DSPs | 1518 | 375 | 25% |
| MLABs | - | 2058 | - |

CPU and GPU implementations are based on the original household FOI algorithm. The CPU reference is evaluated on an Intel Xeon Gold 6230 built on the 10nm process running at 2.10GHz. The GPU software is compiled with NVCC 10.0 and experimented on one Nvidia GeForce RTX 2080 Ti running at 1545MHz. The GPU has 4352 CUDA cores building on the 12nm process. The GPU kernel is developed with block size 256.

### B. Results and discussion

The population size in the experiments ranges from 10 million to 500 million. We record the execution times (T) in milliseconds for the FPGA, 1-core CPU, 10-core CPU, and GPU implementations respectively in columns 2–5 in Table II. In addition to the raw execution times, we present the speed-up (SU) of the FPGA over other implementations in columns 6–8. The implementation on the FPGA platform achieves at most 2.3 (on average 2.2) times speed-up against 10 Intel Xeon Gold 6230 CPU cores, and up to 2.0 (on average 1.9) times speed-up against the Nvidia GeForce RTX 2080 Ti GPU.

TABLE II: Evaluation Result

| Pop. Size | T (ms) FPGA | T (ms) 1C | T (ms) 10C | T (ms) GPU | SU 1C | SU 10C | SU GPU |
|---|---|---|---|---|---|---|---|
| 10M | 76 | 1241 | 174 | 123 | 16.3 | 2.3 | 1.6 |
| 20M | 147 | 2432 | 310 | 261 | 16.5 | 2.1 | 1.8 |
| 30M | 216 | 3717 | 459 | 405 | 17.2 | 2.1 | 1.9 |
| 40M | 287 | 4958 | 627 | 533 | 17.2 | 2.2 | 1.9 |
| 50M | 357 | 6395 | 783 | 684 | 17.9 | 2.2 | 1.9 |
| 60M | 428 | 7449 | 853 | 859 | 17.4 | 2.0 | 2.0 |
| 70M | 498 | 8674 | 1109 | 937 | 17.4 | 2.2 | 1.9 |
| 80M | 570 | 10032 | 1290 | 1129 | 17.6 | 2.3 | 2.0 |
| 90M | 639 | 11178 | 1432 | 1199 | 17.4 | 2.2 | 1.9 |
| 100M | 709 | 13987 | 1613 | 1310 | 19.7 | 2.3 | 1.8 |
| 200M | 1414 | 28566 | 3129 | 2754 | 20.1 | 2.2 | 1.9 |
| 500M | 3588 | 70935 | 7762 | 7192 | 19.7 | 2.2 | 2.0 |

The software on Nvidia GeForce RTX 2080 Ti GPU achieves around 10 times speed-up against single Intel Xeon Gold 6230 CPU core. The main bottleneck for the GPU software is the latency of data transfer for very large population size. It can also be observed that the speed-up number does not change much with the population size, as for very large population, the execution times for all platforms are almost directly proportional to population size. Another main observation is that current bottleneck for FPGA implementation is ALUTs. It can be reasonably estimated that higher throughput and speed-up number can be achieved when there are more bottleneck resources. Besides, there is a non-negligible gap in the speedup over one CPU core. The speedup grows from 17.4 to 19.7 when the population size increase from 90M to 100M. We are not sure about the cause of the gap, but it is probably because the large population size disables some memory optimizations for the CPU code.

## V. Conclusion and future work

Agent-based modeling is a useful decision support tool for epidemics. This paper presents a way to speed up the household FOI evaluation in the CovidSim. Source-level optimizations for the hardware architecture include data caching and control flow adaption. An implementation of the proposed accelerator on an Intel Arria 10GX FPGA achieves 2.2 and 1.9 times speed-up respectively over a CPU reference using 10 cores on an Intel Xeon Gold 6230 CPU and a GPU reference on an Nvidia GeForce RTX 2080 Ti GPU.

A direction of future work is to extend our approach to cover realistic models targeting multiple hardware accelerators, and providing effective support to enable its adoption by epidemiologists. Also, since the simulation time depends on the throughput, we mainly optimize the throughput for the proposed design. A direction of future work is to optimize other aspects, including latency and power consumption.

## Acknowledgement

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] 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.

[2] M. E. Halloran, N. M. Ferguson, S. Eubank, I. M. Longini, D. A. Cummings, B. Lewis, S. Xu, C. Fraser, A. Vullikanti, T. C. Germann, *et al.*, "Modeling targeted layered containment of an influenza pandemic in the United States," *Proceedings of the National Academy of Sciences*, vol. 105, no. 12, pp. 4639–4644, 2008.

[3] N. M. Ferguson, D. A. Cummings, C. Fraser, J. C. Cajka, P. C. Cooley, and D. S. Burke, "Strategies for mitigating an influenza pandemic," *Nature*, vol. 442, no. 7101, pp. 448–452, 2006.

[4] J. M. Carcione, J. E. Santos, C. Bagaini, and J. Ba, "A simulation of a covid-19 epidemic based on a deterministic seir model," *arXiv preprint arXiv:2004.03575*, 2020.

[5] 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.

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

[7] 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.

[8] 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*, pp. 1–12, IEEE, 2008.

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

[10] C. Guo, W. Luk, and S. Weston, "Accelerating simulation for agent-based epidemic models using fpgas," in *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–8, IEEE, 2020.