Scalable Open-Source Reconfigurable Architecture for Bacterial Quorum Sensing Simulations

Alan Ehret, Peter Jamieson† and Michel A. Kinsy
†Miami University, Oxford, OH. jamiespa@miamioh.edu
Adaptive and Secure Computing Systems (ASCS) Laboratory
Boston University, Boston, MA
{ehretaj,mkinsy}@bu.edu

ABSTRACT

Quorum sensing in cells is a generalized framework for modeling and analyzing the local density of the bacterial population in a given biological environment. It has applications in biology, medical and therapeutic domains, e.g., cancer cell research. Software-based simulations are generally slow and only provide a certain level of functional faithfulness or model fidelity. In this work we introduce a scalable open-source architecture to accelerate bacterial quorum sensing simulations called ABAQS (Agent Based Architecture for Quorum sensing Simulation). The presented architecture allows researchers to create and launch new simulations by quickly incorporating custom cell models. The architecture is highly modular and separates the functional model from control logic. It has a simple interface to enable users to readily connect their custom models to the simulation platform. To illustrate the proposed architecture, we present the implementation details and results for a small-scale model representing up to 81 cells which we have synthesized and configured on an FPGA. We also highlight some of the key features to be implemented in future versions of the proposed architecture. The open-source license of this project will allow other researchers to contribute and improve the architecture to (a) better fit their quorum sensing simulations and (b) give the community a flexible simulation acceleration tool.

1. INTRODUCTION

Quorum sensing (QS) is defined as the detection of extracellular chemical signals, which, at certain levels, will alter the behavior of a cell by activating specific genes [9]. By sensing the external concentration of a specific chemical that other cells produce, a cell can infer the number of surrounding cells. QS acts as a form of indirect communication between cells and is sometimes thought of as chemical "wires" in a broadcast communication. In QS communication, cells do not have a mechanism to directly communicate. Instead, any communication or coordination is achieved by sensing chemical molecules outside the cell. Figure 1 depicts several cells in a varying concentration of chemical molecules. The concentration of chemical molecules is proportional to the number of cells nearby.

Researchers are still trying to understand how many cells are needed to reach a quorum and how close cells must be to interact (some studies have suggested distances of no more than 10-100μm) [9]. In addition to researching how and why QS works, recent research has focused on a wide range of potential applications which can leverage QS communication. These applications range from cellular computing, where colonies of bacteria have been grown to compute the output of two input logic gates [8], to targeted drug delivery, where researchers investigate how cells might be engineered to release a treatment in the presence of cancer cells [7]. Previous research has also focused on the role QS plays in biofilm (dense clusters of cells) formation, destruction and resistance to antimicrobial treatments [6]. By understanding how diseases use QS to grow and survive, researchers can develop new therapies that work by interfering with the disease's QS.

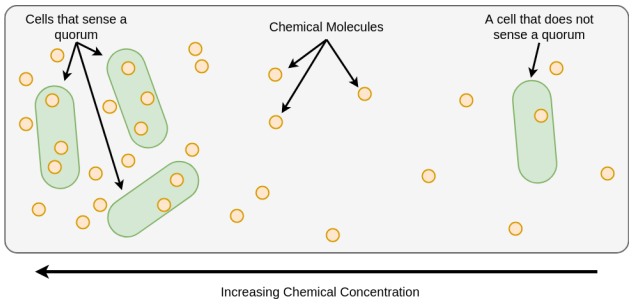

Figure 1: Four cells surrounded by chemical molecules. Each cell is sensing and outputting quantities of the chemical. The cells on the left are in a higher concentration of chemical allowing them to sense a quorum, while the cell on the right is in a lower concentration and does not sense a quorum.

In order to understand how QS systems can be engineered to solve specific problems, researchers need accurate models of these systems and the ability to perform experiments at the relevant scale. Often the scale of the experiments is too large for simulation or laboratory conditions, forcing researchers to use less controllable real world environments. The current QS simulation tools struggle to scale beyond hundreds of micrometers in size. These simulations are large enough to model on the order of 10^5 to 10^6 cells depending on their complexity. While this scale is large enough to simulate the logic gates developed in [8], simulating larger environments will require more scalable simulation tools.

A domain specific architecture would enable researchers to accelerate QS simulations, allowing them to simulate longer time spans at a larger scale than what is feasible in the cur-

Submitted for the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2018) Toronto, Canada, June 20-22, 2018.

rent software frameworks. In this paper, we propose and implement a modular, scalable and open-source architecture to accelerate QS simulations. This open-source architecture serves as a proof of concept and can be further improved by the community. The source code for this project is available on the ASCS Lab website at the following URL: http://ascslab.org/research/abc/abaqs/index.html

2. BACKGROUND AND RELATED WORK

2.1 Quorum Sensing Applications

In the last few years, researchers have begun to understand how QS impacts bacterial interactions by performing experiments in the lab [9] [4]. Some experiments have been performed in more complex natural environments (such as a live organism), but the poor control researchers have over these environments means that the scale and complexity of these experiments is relatively limited.

A current area of research focuses on how diseases use QS. Some works have hypothesized that interfering with a disease's QS could provide a new form of treatment for antimicrobial-resistant infections. These diseases, which are resistant to current treatments, can use QS to coordinate in a way that increases virulence (how harmful a disease is) [9]. Treatments that interfere with QS would not directly treat the disease, instead they would work to reduce the virulence and limit the coordination between cells, potentially slowing or stopping the antimicrobial resistant strains of the disease enough for conventional therapies to be effective.

Another approach which leverages QS is described in [7], where the authors describe their vision of targeted drug delivery, where small doses of a treatment are embedded in nanomachines (cells or other particles) capable of detecting the low oxygen environments preferred by cancer cells. Once these nanomachines detect cancer cells, they would begin to release oxygen. By sensing the concentration of oxygen, the cells can detect when a quorum of nanomachines is present to simultaneously release their treatment. While this type of treatment is likely decades away, it serves as an illustration of the potential uses QS can have in developing new treatments for diseases.

2.2 Quorum Sensing Simulations

Simulation offers a way for researchers to test their understanding of interactions observed in natural and laboratory environments by building mathematical models to represent a cell's behavior. These models can then be used to predict behavior in other environments. Several software frameworks exist to enable researchers to quickly build models of cells and an environment [2] [5]. Most software frameworks take an event driven, agent based approach to model relatively sparse populations of bacteria. This approach allows researchers to simulate events across different time scales with limited processing overhead.

While the agent and event based approaches work well to model individual cells or very small quantities of chemical molecules, chemical concentrations across a 2D or 3D space are better represented with differential equations solved by numerical methods, as in [2]. This combination of chemical fields and agent based cells usually leads to a hybrid simulation where the chemical field is updated at fixed discrete time intervals and the cell states are updated on events in a more continuous time scale.

In [4], the chemical fields and dense cell populations are modeled with differential equations. The diffusion of chemical molecules throughout space is modeled with a discrete Laplace operator shown in Equation 1.

$$x_{t+1} = x_{cell} + \frac{(x_n + x_s + x_e + x_w) - 4 * x_t}{4 * h^2} * dt \qquad (1)$$

In Equation 1, x_{t+1} represents the quantity of chemical in the given location at time $t + 1$. The variable x_t is the quantity of a chemical in given location at time t. The variable x_{cell} is the quantity added by the cell at the same location at time t. The values x_n, x_s, x_e and x_w are the quantities of the chemical in the north, south, east and west locations respectively at time t. The variable h is equal to the physical distance between points in the simulation. The value of dt is equal to the size of the simulation time step. Note that the diffusion model used is meant to represent a 2D space, with unique operations for each edge and corner (not shown in Equation 1).

Most frameworks, such as BSim [2], are written in higher level languages (like Java) which trade off efficiency for usability. These high level languages make it easy for researchers to develop simulations, but these simulations are limited in duration and scale.

2.3 FPGA Accelerated Agent Based Models

Other works have focused on accelerating similar agent based applications with FPGAs. In [1], the author uses memory interleaving to fetch data for a cluster of points in a 2D grid. By taking advantage of the distributed BRAMs on an FPGA, the author is able to get data for a whole cluster in a single memory fetch. Additionally, the author shows how to tile a 2D grid of processing elements, assigning each to a memory bank. The number of banks is dependent on the cluster size needed in the computation. The author of [1] uses Conway's Game of Life as an example agent based application. Conway's Game of Life uses a 3x3 cluster in each computation (a cell and its 8 neighbors), resulting in 9 memory banks. The author shows that they are able to get a 290x speedup compared to a software implementation.

3. PROPOSED ARCHITECTURE

3.1 Processing Element Description

The architecture models a two dimensional discrete space in discrete time. The size of the space is parameterized and user selectable. For simplicity, the edges of the two dimensional space wrap around, creating a torus. This simplifies the implementation by removing the need for unique processing elements along the edges and corners (at the expense of chemical diffusion accuracy with the current model). Conducting simulations with a torus rather than a 2D mesh can help prevent undesirable effects created by the edges of a simulation.

In the current implementation, each point in space is modeled with a single *processing element* (PE) that stores a chemical concentration and the status of a single cell. Separate execution units are used to update the chemical concentration and cell state simultaneously.

Figure 2 shows a block diagram of the architecture. The expanded oval depicts a single PE with a chemical execu-

tion unit (orange box) and a cell execution unit (blue circle). Note that the connections between the cell execution units (dashed lines) are only used to copy state registers in the event of cell movement. Cells cannot directly communicate in QS, instead they must influence the chemical concentration around them which will diffuse to other locations to be sensed by nearby cells. This process is represented with the solid lines in Figure 2.

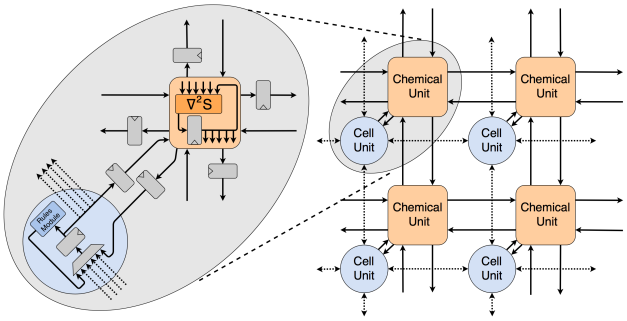

Figure 2: A block diagram of the proposed architecture. The expanded section shows a processing element with chemical and cell execution units and the single word FIFOs between them.

3.2 Cell Execution Unit

The cell execution unit stores the cell's state in a set of registers. Additional registers are used to store the type of cell present in the PE (type 0 is used to indicate that no cell is present). The cell execution unit is designed such that the control logic is separate from the state update logic, allowing a user to quickly make changes to the type of cells being modeled without the need to understand the control signals.

Instead, the user implements a separate *rules module* with pipelined state update logic. The rules module is represented as the dark blue box in the cell execution unit in Figure 2. The interface between the rules module and the rest of the cell execution unit is well defined and constant for each simulation. The control logic makes no assumptions about the data in the state registers, meaning the user is free to use any representation they desire (such as fixed point or floating point) in the custom rules module.

The depth of the pipeline can be determined by the user during the design of the rules module with the parameter RULE_DELAY. User selectable pipeline depths allows for complex cell models without lengthening the critical path or unnecessarily increasing latency.

The rules module will be different for each simulation but it will usually be made up of several pipelines (one for each type of cell) and a multiplexer to choose between them. The type register controls the select signals on the multiplexer, ensuring the correct pipeline updates the state registers. The rules module will also output a signed quantity for the chemical execution unit to add or subtract from the local chemical concentration, representing a cell absorbing or secreting a quantity of chemical. Figure 3 shows a typical cell execution unit implementation with two cell types and a simple 10 or 0 addition to the chemical concentration.

The cell execution unit supports movement of cells in north, south, east or west directions. In order for a cell to move, the rules module must output a direction and assert the move request signal. The control logic in the cell execution unit checks that the adjacent PE is empty. If the PE is empty, the cell is allowed to move, its state registers are copied to the destination PE and the local type registers are set to zero to indicate that the cell unit is now empty. If the neighboring PE already has a cell in it, the control logic ignores the move request and the cell does not move.

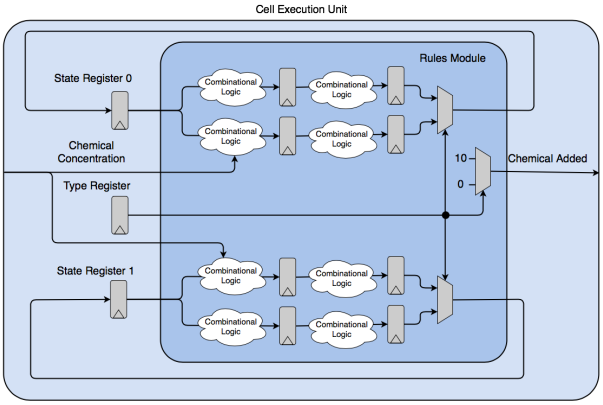

Figure 3: An example cell execution unit with two state registers and two cell types. Note that the chemical concentration input is used in the state update logic. In this example, the cell either secretes 10 or 0 units of chemical based on the type registers.

This implementation has the potential to create deadlocks, as an empty cell execution unit is required for a cell to move. While there are cases where this would be the desired behavior, this may not always be the case. Future work will eliminate this constraint by adding more complicated movement arbitration between cell execution units or by adding several cell execution units to each PE. A PE with several cell execution units can be thought of as modeling a larger two dimensional space with a lower resolution chemical field. A PE with multiple cell execution units would treat all cells as being at the same position in a lower resolution space. Having more cell modules at a discrete point in space would reduce the likelihood of a deadlock. Sacrificing resolution in this way is tolerable because in QS cells do not directly communicate. By placing multiple cells in the same PE, they can be treated as if they are so close to the adjacent cells that they will instantly sense any changes to the chemical environment made by their neighbors.

3.3 Chemical Execution Unit

The chemical execution unit stores the local concentration of chemical at the processing element's location in a register. Each chemical unit is connected to its north, south, east and west neighbors as well as the local cell unit. Two FIFOs are used to connect neighboring execution units, one to buffer data in each direction. The FIFOs are only a single word deep and represented as registers in Figure 2. The discrete time simulation means that any execution unit cannot be more than one time step ahead of its slowest neighbor, resulting in at most one chemical concentration being buffered between execution units. Placing these FIFOs between each neighbor simplifies arbitration and allows a module to update its internal state even if each connected module has not read the previous state yet. The chemical concentration is updated once all of the neighboring modules (including the

cell module) have put new data in the FIFO for the chemical module to read. The model of chemical diffusion is fixed for all simulations but future work will focus on creating modular interfaces for custom chemical diffusion equations, similar to the cell rules modules described above.

3.4 Load and Store Shift Register

To initialize a simulation and record its state, the architecture includes a shift register along each row of PEs. Figure 4 illustrates how these shift registers are connected to the execution units. The execution units can read or write to these shift registers when a global read or write signal is asserted. The length of the shift register depends on the size of the simulation (determined by the user). Each shift register is one word wide (word size is also determined by the user). The shift register length must be long enough to store the state of an entire row at once, so the depth matches the number of state registers for each chemical execution unit (one) and cell execution unit (a user selectable number of registers) in the row. The frequency that the simulation state can be recorded is a function of the number of state registers per PE and PEs per row. A longer row or larger number of state registers will increase the time needed to save the simulation state, reducing how often it can be saved. The architecture uses shift registers along each row, meaning the number of rows does not affect the time needed to record the simulation state but it will determine the bandwidth used to do so.

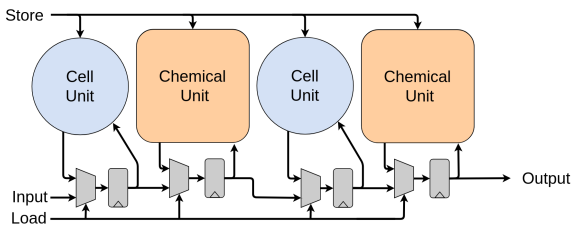

Figure 4: The shift registers used to initialize and record state information. Each row has its own shift register. Global load and store signals are used to read or write state information to shift register.

3.5 Design Flow Overview

Figure 5 shows the process of building a simulation. First, researchers must have a model of their cell. This model can be represented in a variety of formats (equations, software, etc.) and defines the requirements of the rules module. Using the cell model as a guide, researchers can build the Verilog rules module, which typically consists of one or more pipelines multiplexed together by the type register. With the rules module created, the simulation parameters can be set. Parameters include the number of state registers used in the cell model, the depth of the rules pipeline and the size of the simulation. After the parameters have been defined for the specific simulation, the design can be synthesized and configured on an FPGA to record simulation results.

4. RESULTS

To test the architecture, we synthesize it for an Altera Cyclone V FPGA with approximately 32k Adaptive Logic Modules (ALM). Each ALM contains a fractureable 8 input look up table and 4 registers [3]. We use a simple

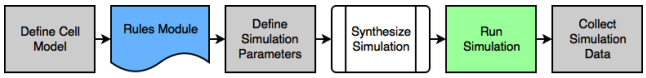

Figure 5: The steps to build a simulation. Designers must build the rules module and define the parameters for the simulation. Each simulation must be synthesized individually. After synthesis, the simulation can be run and results can be saved.

Simulation Size	Number of PEs	ALM Usage	Total Registers	Device Utilization
4x4	16	4876	9767	15%
5x5	25	7533	15229	23%
8x8	64	19190	37906	60%
9x9	81	24277	48436	75%

Table 1: Synthesis results for the proposed architecture with 4 state registers in the cell model.

cell model with 4 state registers. One state register stores the local chemical concentration, another implements a simple counter, the last two store the two previous values of the counter. The word width for each state register (and throughout the architecture) is 32 bits. This model is not meant to model a real cell but instead implement the minimum functionality to verify all of the features of the architecture. This benchmark provides resource utilization results for the simplest models of cells and can act as a baseline resource usage for more complex cell models.

The architecture was synthesized with the parameters described above at various sizes including 4x4, 5x5, 8x8 and 9x9. Table 1 shows the resource usage for each of the sizes.

One PE requires about 300 ALMs. The worst case F_{max} is 71MHz for the 9x9 design. The architecture can compute a simulation time step in 5 cycles, with a single cycle rules module. With 5 clock cycles per time step and a 71MHz clock, a single time step can be computed in 70.4ns. With QS simulation timesteps on the order of milliseconds as in [4], this architecture will enable researchers to quickly conduct simulations with long time spans.

This architecture separates the simulation time step size from the time step between simulation snapshots saved with the row-wise shift registers. This allows simulations to be conducted with a high resolution while ensuring the amount of stored data from the simulation is manageable. The frequency of saved simulation states depends on the length of the shift register and the device's clock frequency.

The 9x9 PE architecture with 1 register in the chemical execution unit for the chemical concentration and 4 registers in the cell execution unit for the cell state has an 45 register shift register. With 5 cycles per simulation timestep, the simulation state can be saved every 9 simulation time steps. Figure 6 plots the number of simulation timesteps between each simulation snapshot for varying numbers of registers per PE.

5. PROJECT ROADMAP

The architecture and results presented are meant to introduce the general framework for using reconfigurable hardware to accelerate QS simulations and to serve as a small scale proof of concept for the platform. Future work will focus on 1) optimizing the architecture for speed and area,

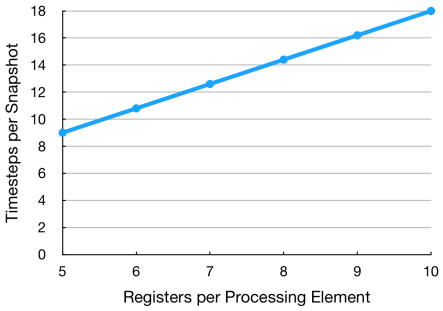

Figure 6: The number of simulation timesteps between saved simulation snapshots. With 5 registers per PE, the simulation can be saved every 9 timesteps. The timestep size can be altered to change the simulation time between saved simulation snapshots.

enabling larger simulations on a single FPGA, 2) implementing a multi-FPGA design, allowing researchers to scale up simulations by adding more FPGAs, 3) improving the architecture's fidelity, enabling more realistic simulations, and 4) creating a programming toolchain for non-digital designers. We plan to release new instances of the platform every six months. The open-source nature of this project will allow researchers to contribute to it, developing the features most important to them.

Features such as time multiplexing the PEs would allow simulations to scale beyond the number of physical PEs in the architecture. A large simulation state could be stored in memory with parts of it loaded onto the PEs to update small sections at a time. Memory interleaving similar to the techniques described in [1] could be used to leverage the FPGA's many BRAMs. The single word FIFOs used to connect neighboring modules work well for discrete time simulations but larger FIFOs would be needed to buffer data in event driven simulations. A parameterized FIFO depth will be incorporated into the next version of the architecture. With the addition of timestamps that can be attached to the data, event driven simulations will be possible. The addition of true multi-word FIFOs between PEs will also separate clock domains, significantly improving the maximum clock frequency of the design. Multi-FPGA implementation support is being developed to allow extremely large and more complex simulations. Future work will be needed to create a communication interface between FPGAs that can efficiently perform the chemical diffusion computation between PEs on separate FPGAs.

The chemical diffusion model is fixed in the current architecture for simplicity but researchers would likely want to tweak it. Placing the diffusion logic in a separate module from the control logic will further improve the flexibility of the architecture. We plan to include this change in the next version of our architecture. Subsequent work will also focus on analyzing the tradeoff between the number of cell execution units per PE and simulation resolution. Multiple cell units on a PE would increase the size of the physical space represented in the simulation at the cost of resolution. Cell movement arbitration must also be improved to avoid deadlocks. We plan to change the arbitration between cells to allow cells to swap places.

To simplify the process of building cell rules modules, higher level tools could be used to describe the pipelines in the rules modules. OpenCL or a C-to-gates synthesizer could be used to describe the cell models in a way more familiar to those already using software frameworks like BSim or Repast to develop QS simulations. Future work could develop toolchains to create cell rules modules from higher level languages. Additionally, a GUI to allow users to quickly define their simulation parameters could simplify the process of simulation building. Frequently used cell rules models could be incorporated into a repository to serve as a starting point for other researchers. These models could be incorporated into a GUI tool to let researchers quickly build simulations with commonly studied cells.

6. CONCLUSION

In this work, we presented a scalable open-source architecture to accelerate bacterial quorum sensing simulations. The architecture is modular and researchers can customize cell models to run different simulations. Furthermore, the control logic is decoupled from the cell model logic to enable researchers to build new cell models without the need to understand the entire architecture. The fact that this project is open-source means that researchers will be able to share the cell models they develop, allowing others to reproduce their results and further build on them. We have presented synthesis results for the architecture with up to 81 processing elements on a single FPGA. Finally, we outlined plans to further improve the architecture's efficiency and efficacy in covering a multitude of cell models.

7. REFERENCES

[1] L. Cui, J. Chen, Y. Hu, J. Xiong, Z. Feng, and L. He. Acceleration of multi-agent simulation on fpgas. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 470–473, Sept 2011.

[2] T. E. Gorochowski, A. Matyjaszkiewicz, T. Todd, N. Oak, K. Kowalska, S. Reid, K. T. Tsaneva-Atanasova, N. J. Savery, C. S. Grierson, and M. di Bernardo. Bsim: An agent-based tool for modeling bacterial populations in systems and synthetic biology. *PLOS ONE*, 7(8):1–9, 08 2012.

[3] Intel. *Cyclone V Device Handbook Volume 1: Device Interfaces and Integration*, 2018.

[4] J. Noorbakhsh, D. J. Schwab, A. E. Sgro, T. Gregor, and P. Mehta. Modeling oscillations and spiral waves in dictyostelium populations. *Phys. Rev. E*, 91, Jun 2015.

[5] M. J. North, N. T. Collier, and J. R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, 16(1):1–25, Jan. 2006.

[6] M. R. Parsek and E. Greenberg. Sociomicrobiology: the connections between quorum sensing and biofilms. *Trends in Microbiology*, 13(1):27 – 33, 2005.

[7] N. R. Raz, M. R. A.-T. *, and M. Tafaghodi. Bioinspired nanonetworks for targeted cancer drug delivery. *IEEE Transactions on NanoBioscience*, 14(8):894–906, Dec 2015.

[8] A. Tamsir, J. Tabor, and C. A. Voigt. Robust multicellular computing using genetically encoded nor gates and chemical wires. 469:212–5, 01 2011.

[9] M. Whiteley, S. P. Diggle, and E. Peter Greenberg. Progress in and promise of bacterial quorum sensing research. 551:313–320, 11 2017.