Towards a Generalized Reconfigurable Agent-Based Architecture: Stock Market Simulation Acceleration

Alan Ehret, Mihailo Isakov, Michel A. Kinsy
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering, Boston University
Email: {ehretaj, mihailo, mkinsy}@bu.edu

***Abstract*—Agent-based simulations have been used in domains ranging from biology to economics. In order to more accurately simulate systems, agent based simulations are becoming larger and more complex. Simulations could benefit from the speed and scale provided by a generalized agent-based architecture. In order to understand which features are relevant for such an architecture, we create a scalable implementation of an agent-based stock market model and compare it to an existing market simulation. The scalable stock market simulation uses several distributed markets rather than a single centralized market. A hardware implementation with support for reconfigurable agent logic and various market topologies is described. Synthesis results for simulations with up to 100 agents are presented. Lessons learned form this domain specific implementation will be applied in future work to a generalized agent-based architecture.**

***Index Terms*—FPGA Simulation, Agent Based Simulation, Stock Market Simulation**

I. Introduction

Agent-based simulations are best used to model systems with many independent entities taking actions in parallel. Systems that have been modeled with agent-based simulations include bacterial colonies, traffic flows, epidemics and stock markets. Agent-based simulations work to model these systems by modeling the individual entities within them. The independent nature of entities such as cells in an organ or cars in a traffic jam means that simulations of these entities should be able to reach similar scales as the systems being modeled. However, most existing agent-based simulations represent a tiny portion of the system they model because of limited compute resources or limited ability to build and analyze such large simulations. The parallel nature of Field Programmable Gate Arrays (FPGA) makes them a good candidate for modeling systems of parallel and independent entities. An FPGA architecture focused on accelerating large scale agent-based simulations could execute such simulations faster and more efficiently than the clusters of CPUs used for today's largest simulations.

Agents in agent-based simulations must be able to sense and influence their environment, taking actions based on their senses independently of other agents [1]. With a population of agents acting in a simulation, a system level behavior will emerge. The system behavior is often more complex than individual agent behavior. Observing this emergent system-level behavior is generally the motivation for conducting a simulation.

Growth in compute resources and improvements in agent-based models have led to an improved understanding of many of these complex systems. Synthetic biologists are beginning to not only understand but also to engineer interactions between cells with the aid of agent-based simulations [2]. Understanding cell to cell interactions has a wide range of potential applications, including treatments for antibiotic resistant diseases. To study the origin of epidemics and how to prevent them, the authors of [3] model populations of wild birds, poultry and people to simulate the spread of avian influenza on a global scale. Researchers in [4] have used agent-based simulations to model individuals trading stocks. Others have built on this stock market model to study how modeling news in the market impacts the model's fidelity [5].

In order to continue advancing our understanding of these complex systems, larger and more detailed simulations are needed. For example, simulations built with the framework in [2] generally model between 10^4 and 10^6 cells but many cell structures (such as organs or whole organisms) have billions or trillions of cells. Similarly, stock market simulations in [4] model just 25 agents and simulations in [6] model less than 10,000 agents while many more people may participate in trading.

As agent-based simulations grow in order to better model a system, they will become increasingly expensive and time consuming to run. General purpose CPUs may not efficiently execute agent based simulations on the scale needed by future researchers. A generalized agent-based simulation architecture designed with a focus on scalability would enable researchers to quickly run large simulations.

There are strong indications that enough features common to nearly all agent-based simulations exist to justify a generalized agent-based architecture [7]. Some of these features include the communication pattern between agents and modeling of the environment. A single generalized architecture is more practical than a series of domain specific architectures as it would support far more domains than dedicated accelerators could be built for. Implementing this architecture on one or more FPGAs would allow researchers to optimize it for their application, implementing agent models in custom logic with a fixed interface to the rest of the system. The ability to customize the architecture will permit researchers to add any necessary features not already in the generalized architecture.

In this work we design the Trading Agent Architecture (TAA), a scalable distributed stock market simulation and implement it on an FPGA to illustrate the design challenges of accelerating scalable agent based simulations. Stock market simulations have been chosen because current software simulations are fundamentally limited in scalability. Lessons learned from this implementation will be applied to a generalized architecture in future work.

II. Related Work

Researchers in [4] implement an agent-based stock market simulation, allowing them to study and understand various aspects of a market. Stock market simulations enable economists to test the impact of different policies or actions in a controlled environment or validate theories about real-world market behavior.

In the simulation, each agent computes their ideal demand or supply given the current market price, the random value of a dividend and their prediction of the next market price. To set a price based on agent demands, a market specialist (MS) collects all of the supply and demand values and determines the difference between the net supply and the net demand. The MS must set a price so that the number of stocks being sold and the number of stocks being bought is equal. The process of collecting demands, setting a price and completing trades is shown in Figure 1. Transmitting the demands from each agent to the single MS limits the number of agents in the system, as more agents means more bandwidth or time is necessary to compute the difference between supply and demand.

978-1-7281-1968-7/18/$31.00 ©2018 IEEE

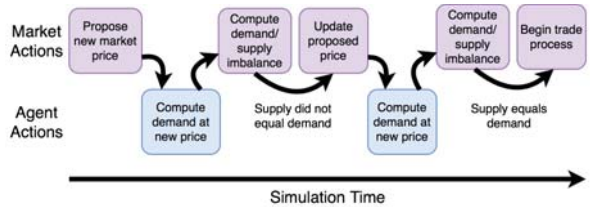

Fig. 1. The process of setting a new market price and executing trades. Local market actions are in purple and agent actions are in blue. Each agent performs the agent actions in parallel.

To compute demands, agents use a set of 100 market predictors. Each predictor has a set of parameters and a market state for which the predictor is considered active. Each trade iteration, the most accurate active predictor is selected to compute an agent's demand. An agent's demand is a function of the current stock and dividend value as well as the selected predictor's parameters and accuracy.

The predictor's market state is represented with a series of true or false conditions. Market states are represented by several true/false comparisons between the current price and the moving average price or the current dividend. Agents use a genetic algorithm (GA) to train their predictions about the next price set by the specialist. The GA updates the parameters as well as the market state for which the predictor is active.

As an agent changes its market predictors to become more accurate, they will change their demand for a given market state. With price being influenced by demand, each agent is trying to make a profit by learning how other agents will value the stock. This prevents agents from reaching an equilibrium where each agent agrees on the value of a stock.

This model of a stock market has served as a baseline for others to extend and experiment with. In [5], news is added to the model to study how incorporating a trader's interpretation of extra information impacts the simulated market's fidelity to real world markets.

The stock market model in [4] has been chosen as the baseline market simulation in this paper because many others have used and extended it. The distributed model proposed here also builds on this popular stock market simulation.

Previous works have tried to improve the scalability of the market model in [4]. Works such as [6] enable larger simulations by distributing agents across several nodes in a cluster. However, agents still communicate with a centralized market specialist which will inevitably limit scaling. The distributed model proposed here overcomes this limitation by removing the centralized market specialist.

III. Model Scalability Analysis

This section analyzes the scalability of each model in a hardware implementation context. To complete a simulation trade iteration in the centralized model described in [4], a market specialist proposes a stock price and broadcasts it to n agents. The agents calculate their individual demands at the given price, and send them back to the market specialist. The market specialist then modifies the stock price to reduce the imbalance between supply and demand. This process is repeated until the supply and demand are equal. Figure 1 illustrates the agent and market specialist actions during this process.

A block diagram of the centralized model is shown in Figure 3. The market specialist broadcasts the stock price over a bus, and agents respond by sending their demands to one of the market specialist input ports. A system with n agents and k market specialist ports must time-multiplex the agent-market communication over n/k time-steps.

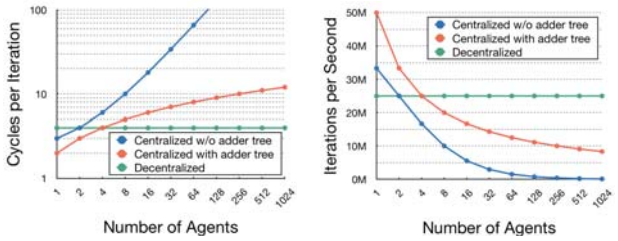

Fig. 2. The number of cycles per price update iteration (a) and the number of price updates per second (b) with respect to the number of agents in the system with a 100MHz clock frequency.

Since the time required for the n agents to communicate their demands to the market specialist grows linearly with the number of agents, this market specialist's bandwidth quickly becomes the bottleneck of the centralized model. A market specialist with several ports trades area for time to sum the agent demands into a net demand in fewer cycles. An adder tree in the market specialist would add up the agent demands in parallel. For n agents, the tree has $n-1$ adders, is $\log_2(n)$ layers deep, with a communication latency of $O(\log_2(n))$ cycles. The tree requires $O(n)$ area to implement. While area grows linearly with the number of agents, the growing latency is still the main bottleneck of the system. As the next price depends on the current price, pipelining cannot hide the communication latency.

Figure 2(a) shows the number of cycles each price update iteration requires with respect to the number of agents for three different systems: (1) the centralized system where the market specialist only has one input port, (2) the centralized system with n input ports where an adder tree sums up the agent demands, and (3) a decentralized system without a global market specialist. The decentralized system is assumed to have 4 agents in each local market and uses 4 input ports with an adder tree. In our implementation, the agents require 2 cycles to compute a demand. With n agents, the single input port centralized system will require $2+n$ cycles, and the centralized system with the adder tree will require $2+\log_2(n)$ cycles to complete a price update iteration. The decentralized system requires a constant 4 cycles per price update iteration (2 for demand calculation and 2 for demand summing by the market specialist). One simulation time-step (one trade iteration) is made up of many price update iterations, as the market specialist must find a price such that supply equals demand.

For the large numbers of agents we are targeting, neither of the centralized models can sustain enough traffic and the whole system quickly becomes communication bound. Figure 2(b) shows the decreasing rate of trades with the increasing number of agents.

Our distributed model does not suffer from the scaling issues of the centralized versions, as the bandwidth needed by local markets is independent from the total number of agents. The system is then able to grow while keeping the number of trade iterations per second fixed. Having a fixed bandwidth between local markets is a desirable feature for large scale simulations. For simulations too large to fit on a single FPGA, the bandwidth needed between FPGAs is determined by the number of agents trading across the boundary between them. Replacing the all-to-one communication with distributed local communication enables the simulation to be designed such that the number of agents trading across two FPGAs is also independent of the simulation size.

In the centralized hardware implementation, many cycles are needed to compute the supply and demand imbalance, preventing efficient scaling of the system. In the distributed system, we remove the all-to-one communication between the agents and the centralized

market specialist. Instead, agents distribute the pricing and demand information through the network of local markets over several price update iterations. While beneficial for scaling, this distribution may make the model unstable when the number of agents is high, as the number of iterations needed to transmit the pricing and demand information will increase. For a torus-based implementation of $n \times n$ agents at least n iterations are needed to transmit information between the most distant nodes. We therefore also test a hypercube-based implementation, which will reduce this communication latency to $\log_2 n$ iterations.

IV. Model Comparison

In order to test the proposed distributed models before building a hardware implementation, agent-based software simulations are used to compare the existing and proposed models.

The centralized model of a market (seen in Figure 3) requires each agent in the market to communicate their supply or demand values to a single market specialist who synchronously makes trades. This all-to-one communication fundamentally limits the size of the simulation. By modeling the market as a series of smaller distributed markets, local market trades become asynchronous and the scalability limitation is avoided. Each local market has its own local price. The local markets in the proposed distributed model are connected by agents participating in several markets at once. The topology of agents and markets can be adapted to best fit the desired simulation conditions. In this section, a checkerboard and a hypercube topology are examined.

In the checkerboard topology, agents (traders) and markets are arranged in a checkerboard pattern such that each agent participates in 4 markets and each market has 4 agents. The edges of the checkerboard are wrapped around, forming a torus. Figure 4 shows the checkerboard topology. A checkerboard pattern was chosen because simulations with less than 4 agents in a market produce drastic price swings because of the difficulty in matching supply to demand. The routing overhead of a hardware implementation of the checkerboard topology will be minimal, as its similarity to a 2D mesh allows it to map well to an FPGA's fabric. If local markets require more than 4 traders, multiple traders could connect two local markets instead of a single trader as is done here. Using multiple traders would enable local markets with an arbitrary (multiple of 4) number of traders.

In addition to the checkerboard topology, a hypercube topology is also examined. In the hypercube topology, each vertex of the hypercube represents a local market with the edges representing agents in 2 markets. The number of dimensions in the hypercube represent the number of agents in each local market. This gives simulation designers a simple way to change the properties of the simulation. Figure 5 depicts the hypercube topology. In the hypercube topology, agents only participate in 2 markets but the number of markets grows slower than the number of agents, meaning that perceived prices have fewer markets to propagate through.

While the distributed model makes some significant changes to the original centralized model, it overcomes scalability challenges related to the all-to-one communication used by a single market specialist. In order to determine how the distributed local markets impact the average global price and trading properties, we run simulations for the centralized, distributed checkerboard and distributed hypercube market models. The trading volume per 10k iterations, mean trader profit and standard deviation of trader profit are shown in Table I for the centralized model and in Table II for the decentralized checkerboard model. Simulations with 4, 16, 25, 36 and 100 agents are compared. The market models used in these simulations are

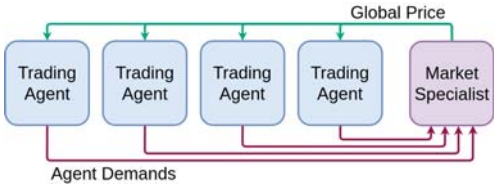

Fig. 3. The communication between the market specialist and trading agents in the centralized model. While the market specialist can broadcast the global price, each agent individually sends their price to the market specialist.

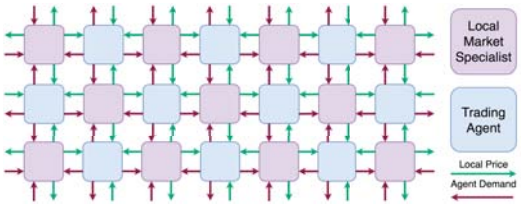

Fig. 4. The distributed model checkerboard topology. Blue boxes are trading agents. Purple boxes are local markets. Each agent is connected to 4 markets and each market is connected to 4 agents.

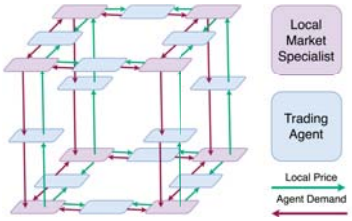

Fig. 5. The distributed model hypercube topology. A 3D hypercube is shown here. Blue boxes are trading agents. Purple boxes are local markets. Each agent is connected to 2 markets and each market is connected to N agents, where N is the dimension of the hypercube.

software implementations of the centralized and proposed distributed market models.

The checkerboard simulations have a higher trade volume because agents are trading on several markets, giving them more opportunities to trade per simulation trade iteration. Although each agent participates in 4 markets, the volume is not four times larger because of the limited money agents have to buy stocks. This prevents a single agent from acquiring a significant portion of the stocks in a market, causing unrealistic market prices. Notice that the volume traded in each decentralized checkerboard simulation is roughly double that of the centralized simulation, indicating that the volume of trades in a single market scales equally in both models.

The centralized model tested here is based on the agent-based simulation in [4] where the mean agent profit is 0.0. Here different simulation parameters cause agents to believe that the stocks are worth more than they would be in an efficient market model. This causes the price in each simulation to range between 80 and 140 when the efficient market model would value the stock around 80 to 100 depending on the value of the dividend. The higher price caused by different parameter settings leads to the consistently negative profit in all simulations. Note that the same market parameters were used in the centralized and distributed model tests, meaning that negative profits are not caused by changes made in the distributed models.

In both models, the mean profit tends to grow with the number of agents. This is likely caused by a higher volume of trading making it easier for agents to buy or sell the ideal amount of the stock. The profits of the distributed model have a lower standard deviation and a flatter upward trend as the number of agents increases. The flatter trend can be expected in the distributed model because agents only interact with a local neighborhood regardless of the number of agents

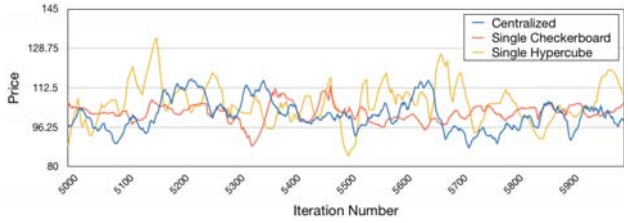

Fig. 6. The market price for the centralized model and a single local market from each of the distributed checkerboard and distributed hypercube models.

in the simulation. This suggests that the agent profits are slow to be influenced by other distant markets, even as more markets are added. Agents still interact with their neighbors though, allowing perceived values to spread, adding some effects from the larger simulation.

In order to test how the topology of the agents and markets in the distributed model impacts the price, volume and profit, another set of simulations are run for the centralized, checkerboard and hypercube topologies. Each simulation has 32 agents. The checkerboard topology has 32 local markets. The hypercube topology uses a 4 dimensional hypercube with 16 markets. Each hypercube market has 4 agents but each agent only participates in 2 markets (compared to 4 in the checkerboard version). Table III compares the volume, mean profit and profit standard deviation for each of the models. As in the other tests, these simulation results have been generated with software implementations of the different market models and topologies.

Note that the volume traded in the hypercube is significantly smaller than in either the checkerboard or the centralized models. This is likely because each agent participates in only 2 small markets, giving it fewer trading opportunities. This comparison shows that the distributed model is sensitive to the topology chosen and that not all topologies will be appropriate for desired volume and price values.

Figure 6 shows the market price in the centralized model and a single local market price from the checkerboard and hypercube models. Only one market from each simulation is shown for clarity. Additionally only 1000 iterations are plotted to make details in the graph visible. In the distributed models, a different random dividend is used for the stock in each local market changing how each agent will perceive the value of the stock. This means that the price time series are not expected to be the same as the original centralized model but should still have similar ranges and properties.

V. The Trading Agent Architecture

a) Architecture Description: The FPGA implementation of the proposed distributed stock market simulation is called the Trading Agent Architecture (TAA). The TAA is designed to enable large scale agent-based simulations of a stock market. These simulations are supported with distributed local markets rather than a single centralized market. A user can change the number of agents or the number of local markets in the simulation by altering Verilog parameters or the simulation topology.

Local market and agent modules are used to model an environment. These modules can be connected in different topologies (such as the checkerboard or hypercube described in Section IV) to model different markets. Local markets support a parameterized number of agents. The fixed interface between each agent and market simplifies the design and creation of simulations with different topologies. The ability to easily connect different processing elements in an arbitrary pattern is a desirable feature in a generalized architecture. The generalized architecture will support communicating a variety of information so that more complex models with features such as news spreading as in [5] can be created and studied.

TABLE I
Mean statistical results for 4 runs of the centralized model.

Number of Agents	Volume per 10k iterations	Mean Profit	Std Dev of Profit
4	1198	-2.99	15.93
16	4726	-2.73	16.29
25	6977	-1.99	10.51
36	9404	-2.40	14.94
100	29158	-0.23	2.33

TABLE II
Average profit and volume statistics for 4 runs of the distributed model with a checkerboard topology.

Number of Agents	Volume per 10k iterations	Mean Profit	Std Dev of Profit
4	2222	-2.50	9.87
16	8392	-2.12	10.27
25	13039	-1.56	9.44
36	16516	-1.57	8.81
100	52798	-1.35	6.44

TABLE III
Comparisons of the centralized, distributed checkerboard and distributed hypercube models. Results shown are the average of 4 simulations with each model.

Model	Volume per 10k Iterations	Mean Profit	Std Dev of Profit
Centralized	8677	-2.51	14.86
Checkerboard	18995	-1.81	8.81
Hypercube	6026	-1.44	4.23

b) The Agent Module: Each agent module computes the demands of a single agent. A separate demand is computed for each market an agent participates in. A market selector in the trade processing logic presents a market price and dividend to the demand computation logic and the computed demand is sent to the selected market. Figure 7 shows a simplified view of the agent module.

A local market will not execute trades until each agent has responded to a trade lock request by asserting the lock acknowledgment signal for the appropriate market. This is necessary as agents will participate in multiple local markets and may not have a new demand ready each cycle. The trade lock signals for each local market allow a user to change the simulation topology or the agent latency without the need to alter the communication protocol.

Agents communicate with the local markets they are connected to with a series of lock request, lock acknowledge and trade valid signals (one for each local market) and dedicated buses for price and demand values. The local market uses a trade lock request signal to indicate that demand and supply are equal and a trade is about to be executed. When the agent is ready to process the trade, it acknowledges the lock request by asserting a lock acknowledge signal. Once all agents in the local market have asserted the acknowledge signal, the local market responds by asserting the trade valid signal indicating that agents should update their cash and position values.

The architecture provides an interface in the agent module that separates user customizable demand computation logic from the trade processing logic needed in all simulations. This interface and customizable logic are represented as a red box in Figure 7. This interface allows new models to be incorporated into the architecture without the need to change the logic handling trade arbitration or cash and position updating. This type of modularity will also be incorporated into the generalized agent-based architecture.

The current implementation of the agent module must store any necessary data for demand computation in on-chip BRAM. There is no interface for agents to access off-chip memory. Off-chip memory is avoided because of the high bandwidth needed during each price

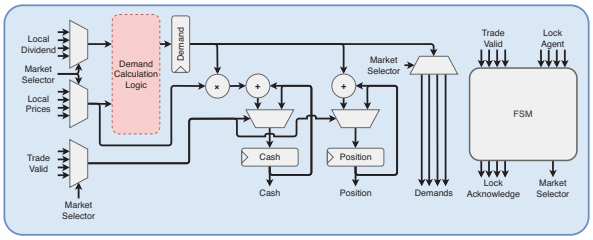

Fig. 7. A simplified view of the agent module architecture.

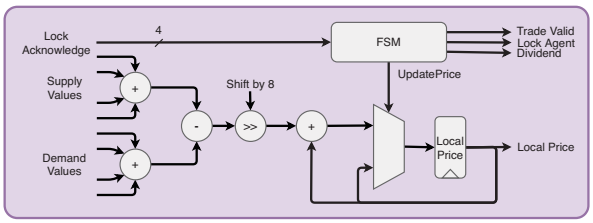

Fig. 8. A simplified view of the local market module architecture.

update iteration, where every agent computes a new demand.

In the software implementation, each agent is randomly initialized. In the hardware implementation, an agent's demand computation memory is initialized to a random valid state. States for each agent are determined by the user at synthesis time. The bit-stream generated by the synthesis tool initializes an agent's BRAM to the given state. Currently, a user must manually determine the appropriate BRAM state for each agent. A tool was not developed to streamline this process because it would be dependent on the agent model used.

c) Local Market Module: The local market module represents the market specialist and is responsible for computing new market price and dividend values. A finite state machine handles the module's control signals. Figure 8 depicts a simplified view of the local market FPGA implementation.

The module takes in supply and demand values from all of the agents in the market and computes the net difference between them. The number of agents in each local market should be small enough that the adder tree described in Section III is practical to implement. The imbalance between supply and demand is shifted by a constant and added to the current proposed local price. Shifting the imbalance replaces costly multiplication (by a power of 2) and creates a value small enough to represent a change in the proposed local price without destabilizing the market. If a change in the proposed price is large, unrealistically wide swings in market price can occur.

While the supply and demand difference is greater than a threshold, the proposed local price is continuously updated and no trading occurs. This is different than the original software model but it avoids the need for division hardware when a maximum number of price update iterations has been reached without a balance between supply and demand, forcing agents to trade at a fraction of their ideal volume.

Once a balance has been reached, the trade process begins. First, the "lock_agent_request" signal is asserted. Once all agents have acknowledged the lock request by asserting the "lock_acknowledge" signal, the local market asserts the "trade_valid" signal. The proposed local price is set as the new local market price and agents update their cash and stock values, completing the trade.

The local market module only needs to store the local price. A new trade imbalance is computed based on demand values set by agents each price update iteration. The small memory requirement of the local market module means that no off-chip memory or on-chip BRAM is needed. The local market price is stored in registers.

d) Market Topology: The local market and agent modules can be connected in different patterns to support different market simulations. The topology of the market could be chosen to model a specific environment, such as traders in different time zones or traders who cannot trade directly.

This paper compares hardware implementations of the checkerboard topology with the original centralized topology. A depiction of the original centralized architecture is shown in Figure 3. The distributed checkerboard topology can be seen in Figure 4.

A checkerboard topology was chosen because simulations with markets containing fewer than 4 agents behaved poorly, creating unrealistic market behavior. In some of these simulations market price would swing wildly by thousands of dollars (rather than tens of dollars). In other simulations, the market prices would diverge, approaching infinity as no buyers for stocks could be found. Synthesis results for the hypercube topology have not been collected because it did not produce a reasonable trading volume for the 32 agent simulation that was tested.

Currently the checkerboard topology is the only supported topology in the TAA. The size of the checkerboard can be adjusted with parameters. The fixed interface between local markets and agent modules means that constructing different topologies only requires the connections in the top level module to be reassigned.

Using several distributed markets allows the TAA to make trades asynchronously. Trades are made when supply and demand are balanced at any local market rather than waiting for global supply to equal global demand. In addition to being a more scalable market simulation, this is a more realistic representation of stock trades. Asynchronous agent interactions with the environment is an important feature that will be included in the generalized architecture.

e) Synthesis Results: Synthesis results for three different implementations are discussed in this section. One implementation simulates the existing centralized stock market model in [4]. The other two implementations use a simpler irrational agent model to simplify topology design and testing. The irrational agent models compute demands with random changes to their parameters rather than calculating demand based on past market history as in the rational model used in the first implementation. A market full of irrational agents is of limited practical use but markets with some irrational agents have been used in [8] to create heterogeneous market predictions similar to the rational agent model implemented here.

In each of the implemented designs, 18 bit fixed point integers are used to represent prices, demands and stock holdings. Each 18 bit fixed point integer uses 9 bits for the integer and 9 bits for the fraction. Cash values have an extra 8 integer bits (for a total of 26 bits) to ensure that they do not overflow. A precision of 18 bits was selected to fit the 18 bit hardware multipliers on the selected FPGA device. Dollar values are not rounded to one cent increments and stocks are not rounded to integer values. The full 9 bits of the fraction is used to maintain as much precision as possible. Fractional stock values can be interpreted as stock splitting between traders or parts of larger bundles of stocks if stocks must be indivisible.

The first architecture implements the centralized market model described in [4]. This version uses rational agents. As few changes as possible were made to the original algorithm for this FPGA implementation. Aspects of the original model including the division needed to trade a fraction of the total requests when supply cannot be balanced with demand after 100 price update iterations are implemented without any hardware specific optimizations. The resource utilization could be improved by restricting several simulation parameters to powers of 2, allowing multiply and division hardware

TABLE IV
SYNTHESIS RESULTS FOR CENTRALIZED SIMULATIONS WITH 4, 8 AND 16 AGENTS WITH RATIONAL AGENT MODELS.

Number of Agents	Logic Elements	Registers	9-Bit Multipliers
4	17724	12249	100
8	25062	14817	196
16	39537	19953	388

TABLE V
SYNTHESIS RESULTS FOR CENTRALIZED SIMULATIONS WITH 4, 16 AND 24 AGENTS WITH IRRATIONAL AGENT MODELS.

Number of Agents	Logic Elements	Registers	9-Bit Multipliers
4	1786	556	12
16	6595	2991	36
24	9741	4431	52

TABLE VI
SYNTHESIS RESULTS FOR DECENTRALIZED SIMULATIONS WITH 4, 16, 24 AND 100 IRRATIONAL AGENTS WITH A CHECKERBOARD TOPOLOGY.

Number of Agents	Logic Elements	Registers	9-Bit Multipliers
4	3470	1080	24
16	13747	4224	96
24	20618	6320	144
100	85760	26232	600

to be replaced with bit shifts. These optimizations were not made to ensure this version was as accurate as possible.

The centralized model, complete with rational agents trained by a GA, is bound by the number of hardware multipliers on a single FPGA. The limited number of multipliers available and the high internal bandwidth of FPGAs means that single FPGA implementations are not significantly limited by the all-to-one communication. However, large simulations across multiple FPGAs, will still be bound by the latency/bandwidth increase shown in Figure 2(a) that comes with a larger number of agents.

The GA logic used to train agents is implemented as a single module that is connected to each agent. The centralized GA logic is less of an issue than the centralized market specialist because agents are only updated by the GA every 250 trade iterations, allowing the accuracy of predictors in agents to be measured between GA updates. This slow rate of GA execution means that the GA has a more relaxed latency requirement than the centralized market authority. For large simulations on multiple FGPAs one GA module can be used per FPGA without creating scaling issues because GA modules only need to communicate with a single local agent at a time, there is no need for GA modules to communicate with logic in other FPGAs.

Table IV shows the resource usage for 4, 8 and 16 centralized rational agent model simulations. Each of the simulations synthesized in this paper target an Altera Cyclone IV FPGA with 150k logic elements. Larger centralized simulations were not synthesized because of the limited number of hardware multipliers on the selected FPGA. The smaller irrational agents allow for larger simulations to be created. More rational agents could be fit on a single FPGA with minor optimizations not expected to impact the simulations fidelity to real markets but these optimizations have not been made in order to create a simulation as close to the software implementation as possible. Additionally, these optimizations would not have enabled the desired scale by themselves. Large simulations will need to be implemented with the distributed model on several FPGAs to maximize the exploitation of parallelism in the system.

As the rational agent model has not been implemented in the distributed simulation, the centralized simulation is re-synthesized with an irrational agent model. This way, the size of distributed and centralized simulations can be directly compared.

The centralized simulation with irrational agents is built with the same agent and local market modules as the distributed version but with a different topology. The topology is changed so that each agent participates in the single centralized market rather than several distributed markets. Table V shows synthesis results for centralized model simulations with 4, 16 and 24 agents. Note that fewer resources are used in the irrational agent simulations because of the simpler agent model.

Table VI shows synthesis results for distributed simulations with 4, 16, 24 and 100 agents. A checkerboard topology is used in each of these distributed simulations. In the checkerboard topology, the number of local markets is equal to the number of agents. The extra local markets add a significant amount of area to the design but also allow for a much larger number of agents by limiting the number of agents in a single market. The extra logic used by local markets will become less significant with larger rational agents which will take up a greater percentage of the total area. The local market module does not contain any hardware multipliers because all multiplication operations have one constant operand that are fixed at a power of 2. This ensures that the number of local markets is not bound by the number of hardware multipliers on a device.

VI. CONCLUSION

In this work, we have discussed the need for a generalized scalable agent-based architecture to accelerate large simulations. Agent-based stock market simulations were examined to demonstrate their limited scalability. A software implementation of a scalable distributed market model is compared to the existing centralized model to examine the trade-offs between the two. To better understand the challenges of creating scalable simulations, we design and implement the Trading Agent Architecture, a distributed stock market simulation. Features of TAA include customizable agent logic and user defined market topologies. The architecture does not achieve the density of agents needed for the large simulations targeted, highlighting the importance of multi-FPGA support in the generalized architecture.

REFERENCES

[1] L. J. Moya and A. Tolk, "Towards a taxonomy of agents and multi-agent systems," in *Proceedings of the 2007 spring simulation multiconference-Volume 2*. Society for Computer Simulation International, 2007, pp. 11–18.

[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*, vol. 7, no. 8, pp. 1–9, 08 2012.

[3] D. M. Rao, A. Chernyakhovsky, and V. Rao, "Modeling and analysis of global epidemiology of avian influenza," *Environmental Modelling & Software*, vol. 24, no. 1, pp. 124–134, 2009.

[4] W. B. Arthur, J. H. Holland, B. LeBaron, R. G. Palmer, and P. Tayler, "Asset pricing under endogenous expectations in an artificial stock market," 1996.

[5] L. Neuberg and K. Bertels, "Heterogeneous trading agents," *Complexity*, vol. 8, no. 5, pp. 28–35, 2003.

[6] C. Wang, C. Yu, H. Wu, X. Chen, Y. Li, and X. Zhang, "A platform for stock market simulation with distributed agent-based modeling," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2014, pp. 164–177.

[7] A. Ehret, P. Jamieson, and M. A. Kinsy, "Scalable open-source reconfigurable architecture for bacterial quorum sensing simulations," in *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, ser. HEART 2018. New York, NY, USA: ACM, 2018, pp. 17:1–17:5.

[8] e. a. De Long, J Bradford, "The survival of noise traders in financial markets," *The Journal of Business*, vol. 64, no. 1, pp. 1–19, 1991.