Leveraging HPC Techniques for Large-Scale Agent-Based Models in Python

Melonie K Richey
George Mason University
Fairfax, VA
melonie.richey@ntconcepts.com

Zachary Mostowsky
Next Tier Concepts, Inc.
Vienna, VA
zachary.mostowsky@ntconcepts.com

ABSTRACT

Agent-based modelling (ABM) and simulation is a domain of social simulation that provides the following benefits: 1) the ability to model individual-level decision-making regimes across many agents, 2), the ability to create spatially-explicit simulations, and 3) the ability to observe simulated behavior at the individual level or in aggregate. As ABMs reach massive scale incorporating many millions, even hundreds of millions of individual agents, model run time and memory quickly become bottlenecks for more robust simulation efforts. ABMs are inherently difficult to parallelize through standard techniques such as multiprocessing and multithreading due to high degrees of inter-agent communication at every time step. This requires that the parallelization techniques maintain separate memory environments for each parallel process and write results back to a master environment, which in itself requires substantial computational overhead. While packages exist to parallelize ABMs in C++, Fortran, and Java, fewer approaches are available in Python and using cloud-based resources. This paper will detail the process of parallelizing a large-scale, spatially explicit, Python-based ABM on Google Cloud Platform (GCP). The use case model is a model of forced migration with refugee agents moving in response to security events worldwide. This paper presents a nearly 3.5X increase in average simulation time step across a variety of migration submodels that include novel stochastic social network elements not included in previous models of forced migration.

ABOUT THE AUTHORS

Melonie Richey is the Advanced Technology Manager at NT Concepts and a PhD candidate in Computational Social Science at George Mason University. Her work involves a variety of programs in the Defense and Intelligence Communities focused on data science and machine learning, deep learning, edge computing, modeling and simulation, and large graph analytics. She holds an MS degree from Mercyhurst University in Intelligence Studies and Methods, and two BA degrees from the University of Florida in Linguistics and Spanish Language. Her technical areas of expertise include analytics, network science, data science, geospatial modeling, and modeling and simulation. She has been involved in modeling and simulation efforts in the domain of migration studies for over five years to include conducting field research abroad on specific migrant communities.

Zachary Mostowsky is a machine learning engineer for NT Concepts working on client-facing programs in the Defense and Intelligence Communities as well as internal R&D initiatives. Mr. Mostowsky attended the University of Maryland where he earned a BS in Geographical Information Systems in 2017 and a MS in Geospatial Intelligence in 2018. His technical areas of expertise include geospatial analysis and architecting cloud native solutions for consuming, processing, and storing data to support model training and prediction. In his free time, Mr. Mostowsky enjoys spending time outdoors, traveling, and attending sporting events.

Leveraging HPC Techniques for Large-Scale Agent-Based Models in Python

Melonie K Richey
George Mason University
Fairfax, VA
melonie.richey@ntconcepts.com

Zachary Mostowsky
Next Tier Concepts, Inc.
Vienna, VA
zachary.mostowsky@ntconcepts.com

INTRODUCTION

The gravity of the global refugee crisis has garnered the attention of researchers, aid workers, and analysts for years. With over 70 million peoples displaced worldwide today, 57 percent of these originate in Syria, Afghanistan, and South Sudan.[1] Other ongoing crises of lesser magnitude but equal tragedy exist in various parts of Central and Southern Africa, South America, and the Middle East. Previous research in this domain has focused largely on voluntary migration undertaken with socioeconomic impetus as opposed to forced migration resulting from threats to personal or physical security. The decision processes and modeling efforts associated with each differ greatly in that the former is a more strategic, long-term process where the latter is reactive, time sensitive, and motivated by very base-level physiological and safety needs.

An Agent-based model (ABM) is the ideal type of model for modeling and predicting the pathways and flows of forced migrants. An ABM is a type of model that allows for multiple agent classes with varying decision-making logic such that patterns in actions and decision-making can be observed on an individual basis and in aggregate. An ABM "allows researchers to specify not only complex characteristics of the environment, but additionally, the complex adaptive 'agents' (i.e. individuals) acting within it. In this type of computational model, the modeling space represents the geography/physical space of the area of displacement." (Edwards 2008) In short, "the focus is on individual agents, their decision processes, their interaction with other agents, and the effects of that interaction on decision processes" (Klabunde & Willekens 2016) ABMs are a natural approach in the computational analysis of forced migration because they allow the simulation of "individual actions of diverse agents [while] measuring the resulting system behavior and outcomes over time." (Crooks et al. 2008) The ability to encode decision-making at the individual level into a model of forced migration is important because migrant decision-making is not random; "their behavior may be unexpected or inconsistent (i.e., noise), but it is not random." (Kennedy 2011) Within the context of the broader modeling and simulation community, ABMs are spatially explicit, temporally explicit, stochastic models. The development of "forced migration models center on patterns of flight" and "the avenue of research has obvious practical implications for governance and humanitarian crisis management." (Frydenlund et al. 2018)

Despite this powerful research potential, the use of ABM for experimenting and exploring geographical phenomena…is still in its infancy," as is the exploration of robust social network analysis techniques in migration networks. (Crooks et al. 2008) Further, "the literature lacks a structured modeling approach to generate agent behavioral rules and initialize agent attributes from data at the individual-level," (Kavak et al. 2018) creating a gap in truly data-driven simulation environments and agent creation. The ABMs that do exist for modeling forced migration and other sociocultural phenomena are also increasing in size and scale, presenting an imminent need to model millions of individual agents over many simulation timesteps. Even with simple model logic and a minimal number of parameters, the computation quickly becomes unwieldly both in memory and processing power required. For this reason, the sociocultural modeling and simulation community is turning to the domain of high-performance computing (HPC) for parallelization methods such as multiprocessing and multithreading.

This paper presents an ABM calibrated to the Syrian refugee crisis and refugee migrants moving from Syria into Turkey in the Spring of 2019. The purpose of the model is to predict likely patterns of refugee movement as refugees move from areas of threat to areas of asylum using upwards of 10 million refugee agents. The model is implemented in Python using standard libraries, and is fully parallelized, containerized, and optimized for cloud-based deployment. This paper is organized as follows: a brief background of forced migration models and previous ABM parallelization efforts, a case study of parallelizing the Syria-based migration model, and finally a discussion of the model parallelization effort.

[1] UNHCR Figures at a Glance https://www.unhcr.org/en-us/figures-at-a-glance.html

BACKGROUND

A small collection of models exists for assessing and predicting voluntary and forced migration flows. "In the past, migration forecasts have been attempted using a wide array of methods, with the central focus frequently on one or more of the following: extrapolation of the past data, the opinion of experts in the field, and the inclusion of additional explanatory information, such as economic data and demographic characteristics." Despite this, "no method is considered to be universally superior, and the applicability of each method depends on the particular definition of migration under scrutiny, as well as the features of the data, such as the length of the series and the stability of trends." (Disney et al. 2015) Traditional methods for modeling and, to a lesser extent, forecasting mass migration have made substantial use of Spatial Interaction Models (SIMs), which include implementations of the gravity model. The gravity model is "an early form of spatial interaction model…which postulates that flows are likely to be, in some way, related to the sizes of the origin and destination areas, and in some way inversely related to distance between them." (Sarra & Signore 2010)

Spatial interaction can take many forms, such as the network- or matrix-based bilateral migrant flow representations found in Tranos et al. (2005) or Rogers et al. (2001). These approaches model and predict spatial interaction using the metrics of network topology such as graph density, cluster coefficients, and latent graph structure. Perhaps one of the most advanced and well-grounded network-based SIMs is Simpop, which represents urban areas as multi-agent systems with nodes (cities) and edges (interactions between urban centers. e.g. the movement of people). (Pumain et al. 1995; Bura et al. 1996; Bretagnolle & Pumain 2010) One of the most robust network-based ABMs addressing forced migration is the FLEE model. (Suleimenova et al. 2017; Suleimenova & Groen 2019; Suleimenova & Groen 2020) This model is also the first of its kind to be fully parallelized in Python and implemented as part of a multi-scale simulation architecture. (Groen 2018; Groen et al. 2019) Other examples of computational models of forced migration include: Hattle et al. 2016, Frydenlund et al. 2018, Parker et al. 2008, and Latek et al. 2013.

Computational Limitations of Agent-Based Modeling

As ABMs reach massive scale, in the many millions to hundreds of millions of simulated agents, computational constraints quickly become prohibitive to verifying, validating, sensitivity testing, and running models. While "most of the existing ABM simulations tend to be small…some models depend on agents with elaborated cognitive models and decision-making processes" that increment the computation required at each iteration exponentially. (Wittek & Campillo 2012) In a serial ABM, the model "iterates through all agents at each time step and has each agent execute some behavior. The time to execute this loop is obviously dependent on the number of agents through which the loop has to iterate and the complexity of the agent behavior." (Collier et al. 2015)

The crux of the computing challenge with ABMs is the following: "The system is intrinsically communication intensive between the various components of the simulation. Agents need to gather knowledge from their environment, as well as from other agents, in order to execute their decision-making processes. Once this phase is completed, there is a possibility that the agents will modify the environment." (Wittek & Campillo 2012) This translates to a need for a shared and dynamic environment that is influenced by every agent at every time step. Notionally, replicating the simulation environment across compute nodes or threads is a straightforward process, but duplicating the simulation environment (in this case, a location graph) "can become a memory bottleneck for extremely large location graphs." (Groen 2018) For models with more than one million but fewer than five million agents, metrics such as three trillion possible agent interactions executing over 60 real-time hours to produce 10 simulated model years with 1-hour time steps have been reported. The original model reported in this research itself took approximately three real-time weeks to simulate a 30-day model run.

Fortunately, there are several techniques which can be implemented within the simulation to alleviate the computational burden. For example, constraining an agent's neighborhood, and draw distance or view distance can reduce the amount of data that must be processed by each agent at each time step. (Segawa et al. 2015) An agent's neighborhood is defined as the cluster of other model agents and model artifacts of which the target agent is aware at any time step. An agent's draw distance represents the distance from the agent's present location from which an agent can gather data for its decision-making function in a model with a spatial component. Simply put, an agent does not always need to be aware of every other agent in the simulation; rather, it may require knowledge of only a few other agents representative of that agent's social network, or agents in the immediate vicinity. Similarly, an agent does not necessarily need to be aware of actions that are occurring in all other spatial localities within the simulation.

High-Performance Computing and Cloud Computing Techniques

HPC is a blanket term applied to the use of advanced computational hardware (such as commercial- and consumer-grade Graphical Processing Units (GPUs)), advanced computational techniques (such as parallelization, multi-processing, and multi-threading), and cloud computing resources to execute "large simulation scenarios containing agents with artificial intelligence algorithms and high computing costs." (Wittek & Campillo 2012) With the proliferation of advanced computational hardware (largely thanks to Nvidia) and the three major cloud companies (Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure) provisioning that hardware in a pay-per-use fashion, research at this scale is now accessible to research domains that require it, e.g. Public Health, Particle Physics, Materials Science, and now, Social Science. Padillo et al. (2014) remind that cloud-based computing "provide[s] all the advantages…to facilitate and democratize modeling and simulation capabilities. Facilitation in terms of allowing computing, storage, and reuse to combine simulation resources while sharing and collaborating use web browsers. Democratization in terms of allowing access across platforms, including mobile platforms and workstations, reducing costs, and being transparent to the final user."

Beyond the hardware the HPC moniker suggests, to is important to distinguish the various types of parallel computing strategies; chief among them multiprocessing and multithreading. When multiprocessing, a task is broken down into component parts and executed in parallel across multiple GPU or CPU cores with separate computational memory and resources allocated to each task. Though the distinction is vague at best, there is some overlap here with distributed computing, whereby tasks are broken down into component parts and sent to separate physical or virtual machines (VMs), sometimes each with their own GPU or CPU clusters. When multithreading, a task is broken down into component parts and executed in parallel across multiple threads on a single CPU core with shared computational memory and resources across the core itself. For the purposes of parallelizing ABMs, multiprocessing is ideal such that each group of agents operates within its own deep copy of the simulation environment, though more sophisticated implementations may involve a combination of the two techniques. (Aaby et al. 2020)

Only a small percentage of ABMs to date have been successfully parallelized and only a handful of those in Python and deployed on cloud infrastructure. The majority of libraries and packages that exist to assist in parallelization efforts exist for C, C++, or FORTRAN. Since Python is one of the most modern and accessible computing languages for the new generation of programmers, especially in the data sciences, it follows that many of the newer and more computationally intensive ABMs are written in this language (or other object-oriented languages such as Java). What lacks, then, and an area of emerging research, is a suite of libraries, packages, tools, and techniques to parallelize ABMs written in Python and deploy on cloud infrastructure. A survey of current HPC software for ABMs is available in Groen et al. 2019, though by far the most pervasive within the CSS community is Repast HPC. (Collier & North 2012) Specifically for parallelizing in Python, there is MESA, an open-source ABM framework with the goal of "provid[ing] a Python alternative to other agent-based modeling frameworks such as NetLogo and Mason." (Masad & Kazil 2015)

The process for parallelizing ABMs mostly involves "refactoring models themselves to work in multi-process non-shared memory environments found in clusters and high-performance computers. In the multi-process parallel environment, each process is responsible for agents *local* to that process. That process executes code that represents the agents' behavior. Data (e.g. agents) can be shared across processes using Message Passing Interface (MPI)." (Collier et al. 2015)

USE CASE: AN ABM OF FORCED MIGRATION IN SYRIA

Agent Decision-Making and Model Logic

The overall purpose of the model is to represent and predict refugee and Internally Displaced Person (IDP) movement patterns across geographic terrain when forced migration events occur worldwide. Specifically, the model addresses the following research question: 1) Given spatial and contextual input data, to where are refugee populations likely to move during and directly following a forced migration event?

A single agent in the base model is representative of a forced migrant refugee. Each agent initializes with a stochastic social network consisting of between zero and three kinship ties (i.e. family members) and zero and three friendship ties (i.e. acquaintances). At each time step, agents move between locations in the simulation environment based on social and environmental factors. The simulation environment is a network of location nodes with links between those location nodes that share an administrative border (See Figure 1). The data preprocessing pipeline creates a weighted network architecture from the following input data:

- The UNHCR's current locations of refugee camps and open border crossings during an ongoing security crisis or event (UNHCR n.d.)
- The Armed Conflict Location & Event Data Project (ACLED) dataset for the appropriate time period filtered down to include only violent protest, terrorist, or other security-related events that occur within the simulation spatial extent and temporal range (Raleigh et al. 2010)
- The UNHCR's (or other source's) existing refugee population by some administrative level above administrative level 0 (UNHCR n.d.)
- Open source shapefiles representing the spatial extent of the simulation environment at the desired level of granularity for the simulation (in this case, administrative level 2)

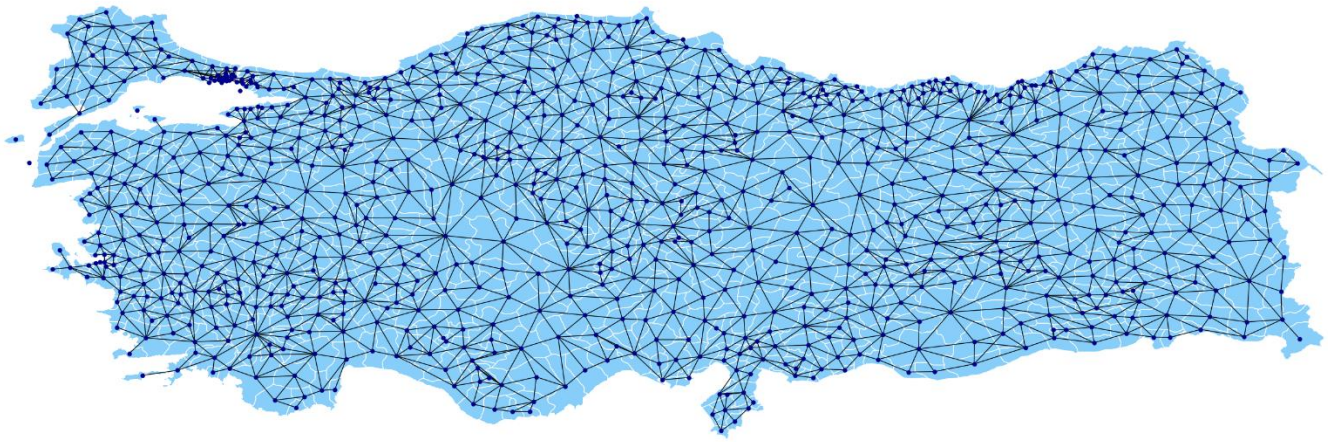

Figure 1. Spatial extent of model simulation environment (Turkey, administrative level 2) with 947 location nodes.

At every time step of the simulation (representative of one calendar day), each agent probabilistically determines whether or not to move from its current location to the candidate location nodes within its view range. Each agent then calculates a location node desirability score for each candidate location node based on static environmental factors (such as presence of refugee camps, existing refugee population, and conflict events affecting physical security) and stochastic social factors (such as presence of friends or kin). The location node with the highest desirability score is the node to which the refugee agent moves at that time step. This calculation is made for all 4 million agents in the simulation at every time step and then the simulation environment network is updated with current refugee agent locations. A more detailed description of the model can be found in Richey (2020, *in press*).

Parallelization Techniques

The base model implementation is parallelized by batching refugees at every time step using a modifiable model parameter. Notionally, the number of batches can correspond to the number of CPU cores on the provisioned virtual or physical simulation machine. An optimal batch size can be found through hyperparameter testing. By drip feeding processes with refugees, we assure that no one process gets hung up processing a large batch of refugees while the other processes sit idle. The downside of this is the large overhead in communicating new data to the processes.

Once the number of parallel processes has been established, each process receives a tuple containing start and stop indices indicating the refugees for which each process is responsible. Each process then works independently on its batch of refugees, pulling the refugees from shared memory and aggregating the results up to the main process after the parallel computation is complete. By leveraging multiple processes, we attain a nearly 3.5X speed increase across experimental conditions using this method. The model parallelization and logic processes that execute at each time step are:

1. **Calculate**: Calculate or re-calculate deterministic portion of location score
2. **Split**: Calculate indices to batch migrant agents and into *x* number of groups for decision-making parallelization
3. **Move:** Execute agent decision-making and movement. For each migrant agent in group:
    1. **Activate**: Decide whether or not to move from current location in accordance with *percent move chance* probabilities.
    2. **Gather**: Fetch neighboring location list as a list of candidate move locations

3. **Assess**: Compute *desirability scores* for each candidate location based on environmental (location) and social (agent) state variables
4. **Move**: Create new agent at location with highest desirability score.
5. **Return**: From each process, return a list of new refugee objects and a dictionary to store the refugees at each node after moving

4. **Update:** Aggregate refugees from each process and overwrite refugee objects from previous timestep. Aggregate dictionaries of refugee locations. Calculate new weights from aggregated dictionaries and updated location entities in the graph.
5. **Create:** Create relationships between co-located refugees.

Parallelization Tests

The tests in this section establish a baseline and identify thresholds for various parameters of the ABM simulation. These include the effect the size and complexity of the physical simulation environment, number of refugee agents, and average size of a refugee's social network have on simulation step time. Additional tests measure the impact of parallel processing on variable size social network simulations. The results of all tests are averaged over 10 time steps using 1,000 location nodes with an average of 5 location neighbors and 100,000 agents with static social networks containing 1 kin and 1 friend unless otherwise indicated. All tests are run on a virtual machine with 104GB of memory, 16 CPU cores, and use 4 parallel processes in Google Cloud Platform (GCP). For reference, an average full model run requires 30-90 time steps.

Table 1. Model step run times (in seconds) based on several structural factors

# Location Nodes (static social network)	Time per step	**# Location Nodes (dynamic social network)**	Time per step	**Avg. # Location Node Neighbors (10k nodes)**	Time per step	**# Agents**	Time per step
10	3.68	10	5.52	1	2.47	100	0.13
50	3.75	50	5.62	5	2.55	1,000	0.13
100	3.87	100	5.76	50	3.41	10,000	0.42
500	4.46	500	6.24	100	4.51	100,000	3.42
1,000	5.00	1,000	6.91	1,000	26.41	1,000,000	31.86
10,000	17.43	10,000	19.25	5,000	144.66	10,000,000	330.96
100,000	243.67	100,000	243.79	*10000	287.45	20,000,000	nd

Several tests were run to get a sense for how various structural elements of the model affect model run time. Table 1 presents the model step run time in seconds of the model with size variations. The following structural elements were tested: effect number of simulation environment location nodes has on step time (refugees have a static social network with 2 ties), effect number of simulation environment location nodes has on step time (refugees have a dynamic social network with between 0 and 6 ties), effect average simulation environment location graph density has on step time, and effect number of total model refugees has on step time. From these results, we make two observations. First, model step time increases linearly with the number of refugee agents in the model, averaging 30 seconds per million refugees while using 4 parallel processes. By contrast, serial processing with the same parameters averages over 10 minutes per million refugees using only a small, static social network for each agent. Second, it is number of agents in the network and number of social ties that most impact model step time, not size of the simulation environment or simulation environment location graph density.

Table 2. Scalability results from representative model runs.

# Agents	# Social Ties	# Processes	Time to Completion	Speedup
1M	2	1	34.13	1.00
1M	2	2	28.42	1.20
1M	2	4	19.78	1.72
1M	2	8	16.04	2.12
1M	2	12	16.40	2.08
1M	2	16	16.62	2.05
5M	2	1	181.50	1.00
5M	2	2	149.73	1.21
5M	2	4	104.53	1.73
5M	2	8	84.25	2.15
5M	2	12	83.60	2.17
5M	2	16	86.02	2.10
10M	2	1	347.60	1.00
10M	2	2	287.02	1.21
10M	2	4	189.76	1.83
10M	2	8	138.70	2.50
10M	**2**	**12**	**124.40**	**2.79**
10M	2	16	152.91	2.27
25M	2	1	918.89	1.00
25M	2	2	726.88	1.26
25M	**2**	**4**	**461.06**	**1.99**
25M	2	8	NA	NA
25M	2	12	NA	NA
25M	2	16	NA	NA
1M	50	1	249.61	1.00
1M	50	2	172.68	1.44
1M	50	4	106.55	2.34
1M	50	8	76.26	3.27
1M	50	12	72.26	3.45
1M	**50**	**16**	**70.46**	**3.54**

Parallel processing tests were run using 1 million, 5 million, 10 million, and 25 million refugee agents comparatively, with a simulation environment size of approximately 1,000 nodes and a static social network size of 2 ties per agent (See Table 2). With these conditions, we observe up to a 2.8x speedup in total simulation step time using 12 parallel processes and a 2x speedup using only 4 processes. In the final parallel test condition, we use 10 million refugee agents with a social network size of 50 ties per agent, which, using serial processing, would slow the simulation to unusable levels. In this condition, we observe a 3.5x speedup in total simulation step time. In even the most computationally intensive submodel conditions (i.e.

millions of refugees with robust social networks of 50 other agents), speedup is significant, improves upon industry standard migration model speedup, and increases steadily with the more parallel processes that are used.

DISCUSSION

Test results indicate a speed performance increase across all tested model conditions while multiprocessing with a batch size of between 2 and 16 parallel processes. Up to 6 parallel processes are typically available on a consumer-grade computing machine where up to 16+ parallel processes are possible when provisioned on VMs in cloud environments. This holds true for submodels with large volumes of refugee agents, submodels where refugee agents have larger social networks, and submodels with larger simulation environments than that currently represented in the base model. Speed, however, does not increase linearly for each process as the overhead time spent on inter-process communication (IPC) at some point counteracts the speed increase in calculation time. IPC time includes the time to copy each sim environment to each process and to write results back to the main process. It is a relatively straightforward to parallelize an ABM with a few thousand agents, but as the number of agents, location nodes, and average graph density (i.e. number of edges) increases, the memory requirements to store the model parameters becomes very large (10s – 50s of GBs). For each process across which computation is divided, each requires an individual copy of the simulation environment or, at the very least, access to variables that determine the dynamic portion of the node desirability score (i.e. the social variables). For this reason, the larger the simulation, the fewer processes can be used because as the simulation grows, it takes longer to make the simulation environment copy to the child processes. Eventually, the overhead is more than the speed gain and the constraint will be total machine memory.

Groen (2018) reports speed increases ranging from 2.5x to 3.4x using up to four parallel processes for the FLEE model of forced migration. (Suleimenova et al. 2017) This paper concludes a 2x to 3.5x speed increase across several experimental conditions. This model also instantiates stochastic elements not found in the FLEE model, most notably a dynamic social network component. As shown in the initial experiments, this is the most computationally intensive component of the model and therefore places the highest demands on parallel computing processes.

The speed increases reported in this paper do not sacrifice model accuracy for the sake of speed. Model accuracy is being incrementally approved and researched simultaneously. Future submodels will address the social science-based formulation of the model logic which, especially when dealing with the social network, may increase step time and increase computational intensity. As this work continues, improvements to model accuracy will be made with continued advances in computation in mind. For example, future iterations of this ABM will include dynamically updating the social network at each time step increasing the stochasticity of the model even further. To accomplish this, lists of refugees will also be stored as a property of each node. This will allow the dynamic creation of friendship and kinship ties between co-located agents at each time step. Additionally, storing refugee lists as a property of each node will cut down on IPC time as only the refugees to be processed in that batch will need to be passed to the process along with the geophysical graph. While lookups to determine if friend or kin are at a neighbor node will take longer, the shortened IPC time and memory requirements should allow for a greater degree of parallelization, effectively reducing step time. Further experimentation should also be conducted to determine the average time a single refugee agent takes to process. A very large range suggests drip feeding processes with fewer refugees while a consistent processing time could allow for very large batches to be sent to each process at once.

REFERENCES

Aaby, B. G., Perumalla, K. S., & Seal, S. K. (2010, March). Efficient simulation of agent-based models on multi-GPU and multi-core clusters. In *Proceedings of the 3rd international icst conference on simulation tools and techniques* (pp. 1-10).

Bretagnolle, A., & Pumain, D. (2010). Simulating urban networks through multiscalar space-time dynamics: Europe and the united states, 17th-20th centuries. *Urban Studies*, *47*(13), 2819-2839.

Bura, S., Guérin-Pace, F., Mathian, H., Pumain, D., & Sanders, L. (1996). Multiagent systems and the dynamics of a settlement system. *Geographical analysis*, *28*(2), 161-178.

Collier, N., & North, M. (2012). Repast HPC: A platform for large-scale agent-based modeling. *Large-Scale Computing*, 81-109.

Collier, N., Ozik, J., & Macal, C. M. (2015, August). Large-scale agent-based modeling with repast HPC: A case study in parallelizing an agent-based model. In *European Conference on Parallel Processing* (pp. 454-465). Springer, Cham.

Crooks, A., Castle, C., & Batty, M. (2008). Key challenges in agent-based modelling for geo-spatial simulation. Computers, Environment and Urban Systems, 32(6), 417-430.

Disney, G., Wiśniowski, A., Forster, J. J., Smith, P. W., & Bijak, J. (2015). Evaluation of existing migration forecasting methods and models. Report for the Migration Advisory Committee: Commissioned research. ESRC Centre for Population Change, University of Southampton.

Frydenlund, E., Foytik, P., Padilla, J. J., & Ouattara, A. (2018, December). Where are they headed next?: modeling emergent displaced camps in the DRC using agent-based models. In *Proceedings of the 2018 Winter Simulation Conference* (pp. 22-32). IEEE Press.

Groen, D. (2018, June). Development of a multiscale simulation approach for forced migration. In *International Conference on Computational Science* (pp. 869-875). Springer, Cham.

Groen, D., Knap, J., Neumann, P., Suleimenova, D., Veen, L., & Leiter, K. (2019). Mastering the scales: a survey on the benefits of multiscale computing software. *Philosophical Transactions of the Royal Society A*, *377*(2142), 20180147.

Hattle, A., Yang, K. S., & Zeng, S. (2016). Modeling the Syrian Refugee Crisis with Agents and Systems. UMAP Journal, 37(2).

Kavak, H., Padilla, J. J., Lynch, C. J., & Diallo, S. Y. (2018, April). Big data, agents, and machine learning: towards a data-driven agent-based modeling approach. In *Proceedings of the Annual Simulation Symposium* (p. 12). Society for Computer Simulation International.

Kennedy, W. G. (2012). Modelling human behaviour in agent-based models. In *Agent-based models of geographical systems* (pp. 167-179). Springer, Dordrecht.

Klabunde, A., & Willekens, F. (2016). Decision-making in agent-based models of migration: state of the art and challenges. *European Journal of Population*, *32*(1), 73-97.

Łatek, M. M., Rizi, S. M. M., & Geller, A. (2013, December). Verification through calibration: an approach and a case study of a model of conflict in Syria. In 2013 Winter Simulations Conference (WSC) (pp. 1649-1660). IEEE.

Masad, D., & Kazil, J. (2015, July). MESA: an agent-based modeling framework. In *14th PYTHON in Science Conference* (pp. 53-60).

Padillo, J., Diallo, S., Barraco, A., Kavak, H., & Lynch, C. (2014). Cloud-based simulators: Making simulations accessible to non-experts and experts alike. Proceedings - Winter Simulation Conference. 2015. 10.1109/WSC.2014.7020192.

Parker, D. C., Entwisle, B., Rindfuss, R. R., Vanwey, L. K., Manson, S. M., Moran, E., ... & Mussavi Rizi, S. M. (2008). Case studies, cross-site comparisons, and the challenge of generalization: comparing agent-based models of land-use change in frontier regions. Journal of Land Use Science, 3(1), 41-72.

Pumain, D., Sanders, L., Mathian, H., Guérin-Pace, F., & Bura, S. (1995). SIMPOP, A Multi-Agents Model for the Urban Transition.

Raleigh, C., Linke, A., Hegre, H., & Karlsen, J. (2010). Introducing ACLED: an armed conflict location and event dataset: special data feature. *Journal of peace research*, *47*(5), 651-660. https://acleddata.com/#/dashboard

Richey, M. K. (2020, *in press*). Modeling Forced Migrant Populations Using ABMs. Proceedings – SBP-BRiMS International Conference on Social Computing, Behavioral Cultural Modeling, and Prediction and Behavior Representation in Modeling and Simulation. 2020.

Rogers, A., Willekens, F., Little, J., & Raymer, J. (2002). Describing migration spatial structure. *Papers in Regional Science*, *81*(1), 29-48.

Sarra, A. L., & Del Signore, M. (2010). A dynamic origin-constrained spatial interaction model applied to Poland's inter-provincial migration. *Spatial Economic Analysis*, *5*(1), 29-41.

Segawa, S., Kin, S., Kawamura, H., & Suzuki, K. Implementation of Massive Agent Model Using Repast HPC and GPU.

Suleimenova, D., Bell, D., & Groen, D. (2017). A generalized simulation development approach for predicting refugee destinations. Scientific reports, 7(1), 13377.

Suleimenova, D., & Groen, D. (2019). How policy decisions affect refugee journeys in South Sudan: a study using automated ensemble simulations. (*in press*)

Suleimenova, D., & Groen, D. (2020). How policy decisions affect refugee journeys in South Sudan: a study using automated ensemble simulations. *Journal of Artificial Societies and Social Simulation*, *23*(1), 1-2.

Tranos, E., Gheasi, M., & Nijkamp, P. (2015). International migration: a global complex network. Environment and Planning B: Planning and Design, 42(1), 4-22.

United Nations High Commissioner for Refugees (UNHCR). (n.d.). Data Sources. Retrieved February 1, 2019, from https://www.unhcr.org/en-us/data.html

Wittek, P., & Rubio-Campillo, X. (2012, December). Scalable agent-based modelling with cloud HPC resources for social simulations. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings* (pp. 355-362). IEEE.