Software Technology to Develop Large-Scale Self-Adaptive Systems: Accelerating Agent-Based Models and Fuzzy Cognitive Maps via CUDA

Kareem A. Ghumrawi, Kim Ha, Jack T. Beerman, John-David Rudie, Philippe J. Giabbanelli
Department of Computer Science & Software Engineering
Miami University
Oxford OH, USA
{ghumraka,hatk,beermajt,rudiejd,giabbapj}@miamioh.edu

Abstract

Agent-Based Models (ABMs) have long served to study self-adaptive systems and the emergence of population-wide patterns from simple rules applied to individuals. Recently, the rules for each agent have been expressed using a Fuzzy Cognitive Map (FCM), which is elicited from a subject-matter expert. This provides a transparent and participatory process to externalize the 'mental model' of an expert and directly embed it into agents. However, software technology has been lacking to support such hybrid ABM/FCM models at scale, which has drastically limited the scope of applications and the ability of researchers to study emergent phenomena over large populations. In this paper, we designed and implemented the first open-source library that automatically accelerates ABM/FCM models by leveraging the CUDA cores available in a Graphical Processing Unit. We demonstrate the correctness and scaling of our library on a case study as well as across different networks representing agent interactions.

1. Introduction

Agent-Based Modeling (ABM) is one of the most valuable techniques to develop self-adaptive systems (SAS) as they support key aspects such as dynamics and decentralized control, emergence and self-organization [Krupitzer et al., 2015]. Studies have applied ABM to SAS for decades, with examples illustrating how ABM can support runtime adaptation in software systems [Qureshi and Perini, 2008] or how SAS could be achieved at a massive scale [Cambier et al., 2002]. The ABM approach uses individual entities (*agents*) that act autonomously and can interact with other agents (e.g., to coordinate and collaborate) and the environment. The *simple rules* enacted at the individual level allow to observe emergence in population-wide patterns, as illustrated over the many cells of an organism or the population of an entire country [Barde and Van Der Hoog, 2017, Li and Giabbanelli, 2021]. A large number of Agent-Based Model toolkits are available [Abar et al., 2017] and some are able to operate with large-scale populations (e.g., `Repast`).

Although manually crafting and adjusting the rules governing the behaviors of agents based on theories and data has enjoyed many successes and remains a common approach, there are several potential drawbacks. First, transparently explaining and refining the rules with subject-matter experts can be a challenge [Voinov et al., 2018], particularly for interdisciplinary problems. Second, there can be issues of replicability, as the model-building heuristics of a team [Freund and Giabbanelli, 2021] may only be implicit or driven by experiences more than data and theories. Third, agents created through this process may adequately produce the emergent properties used for validation in a given application domain, but their individual-level rules may be partly disconnected from reality. In reaction to these limitations, a framework was previously proposed to transparently elicit the rules governing an agent from subject-matter experts. That is, the 'mental model' of an expert is elicited and represented as a *Fuzzy Cognitive Map* (Figure 1), then embedded into an agent to govern its decision-making processes [Giabbanelli et al., 2017, Mkhitaryan and Giabbanelli, 2021]. Such ABM/FCM hybrid models (Figure 2) have been employed in several participatory modeling studies to transparently create a holistic picture of agents with heterogeneous behavior rules influencing each other [Davis et al., 2019]. However, hybrid ABM/FCM models have thus far been limited to simulating small populations. Despite the existence of frameworks to achieve scaling by running an ABM in parallel [Abar et al., 2017] or executing FCMs in parallel [Lavin and Giabbanelli, 2017], the complexity of hybrid ABM/FCM models has resulted in software that run simulations serially [Giabbanelli et al., 2019] and hence cannot cope with the large-scale population sizes required in many studies (Table 1).

URI: https://hdl.handle.net/10125/103465
978-0-9981331-6-4

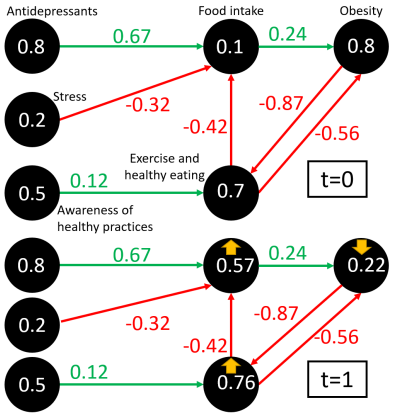

Figure 1: A Fuzzy Cognitive Map (FCM) represents concepts as nodes and their causal impact as directed, weighted edges (that may form loops). Node values are iteratively updated until key concepts stabilize.

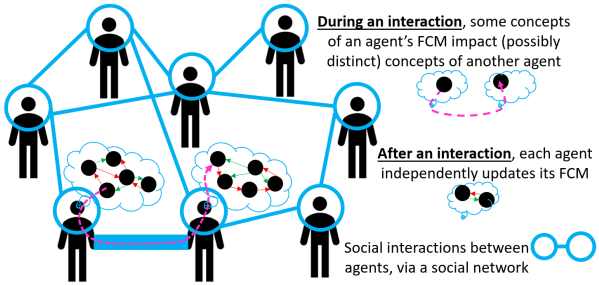

Figure 2: In a hybrid ABM/FCM model, each agent has its own FCM, with a potentially different structure. FCM concepts correspond to an agent's attributes. A simulation repeatedly gets agents to interact (thus giving an impetus to parts of their FCMs) and reflect (thus independently updating their FCMs).

In this paper, we develop a new library that modelers can use to create large-scale self-adaptive systems under the ABM/FCM approach. Specifically, we accelerate ABM/FCM simulations by parallelizing them automatically for modelers using a General-Purpose *Graphics Processing Unit* (GPGPU) via Nvidia's proprietary `CUDA` library. On a distributed architecture such as GPGPU, subsets of independent agents can be simulated simultaneously, causing the simulation to run quicker overall [Tang and Bennett, 2011]. By playing to the strengths of the GPGPU's highly parallel computing environment, which allows us to run more threads than a traditional *Central Processing Unit* (CPU), our library can significantly decrease the clock time of a hybrid model without requiring specialized knowledge from users. Although the execution of ABMs on the GPU using CUDA is not new [Richmond et al., 2010, Hesam et al., 2021], our work is the first that uses CUDA to accelerate ABM/FCM hybrid models, in contrast to previous platforms that had a strictly serial execution and relied exclusively on the CPU [Giabbanelli et al., 2017].

Table 1: Hybrid ABM/FCM models are typically small, that is, they support only a few thousand agents.

Sizes of Hybrid Models	
Reference	ABM Size
[Abdou et al., 2012]	2000
[Giabbanelli et al., 2014]	$\leq$ 2412
[Grantham and Giabbanelli, 2020]	1800
[Mehryar et al., 2019]	154
[Stula et al., 2010]	4

To facilitate its deployment, our `cuda-hybrid` library is hosted on the Python Package Index (`https://pypi.org/project/cuda-hybrid/`) and its documentation (including tutorials) is accessible at `https://cuda-hybrid.github.io/`.

Our overarching contribution is to propose the first library that automatically accelerates hybrid ABM/FCM models for modelers using CUDA. This contribution is realized through three specific aims in this paper:

1. We introduce the *design and implementation* of our library, focusing on the high-level logic that guides modelers in executing their own projects.
2. We use a *case study to show the correctness* of the library, by replicating the emerging results of a previously published ABM/FCM model running in the traditional serial manner.
3. We *demonstrate that massive scalability is achieved* via the library, but that the exact runtime of a scaled simulation can be affected by the structure of the interactions between agents.

The remainder of this paper is organized as follows. In Section 2, we succinctly cover the creation and acceleration of traditional ABMs then briefly summarize the principles of hybrid ABM/FCM models. We introduce our library in Section 3. A case study is presented in Section 4 with accompanying results in Section 5 to demonstrate the correctness and scalability of the library. The significance of these results are discussed along with the current limitations in Section 6. Lastly, we summarize our work and detail the importance of running a large population on hybrid models on the GPU in Section 7.

2. Background

2.1. Creating and accelerating an ABM

An ABM is composed of *agents* (type, attributes, actions), an environment, and a set of rules. These rules control individuals and help replicate behaviors, actions, and measures that resemble real-world phenomena at a larger unit (e.g., group or population-wide). For example, the Schelling Segregation Model defines each agent as an ethnic or social group/area and sets rules on how agents relocate based on their neighbors [Grantham and Giabbanelli, 2020]; racial clusters may form as a result of individual preferences.

Interactions between agents can be modeled via a network, where a node represents an agent and an edge represents a connection between two agents. Networks may possess certain properties. *Scale-free networks* are characterized by a power-law degree distribution, which means that a minority of agents have many peers, and newer agents are likely to further grow the social capital of these highly social agents. *Small-world networks* have groups of agents (creating a high global clustering coefficient) and a few agents from different groups interact, leading to 'shortcuts' across the network (yielding a low average path length). Generators can create networks with these target properties [Amblard et al., 2015].

Large scale parallel and distributed simulations are commonplace for ABMs [Taylor, 2019]. The problem is often studied as a matter of *partitioning* a population of agents within a population of servers (which broadly include threads or compute nodes depending on the level of parallelism), while ensuring a similar *workload* across servers (so that a handful of servers are not delaying the simulation while others are idle) and minimizing *communication costs* between them. This problem is NP-Complete [Lui and Chan, 2002] hence there is a large number of heuristics. Historically, the design of these heuristics has tended to cover three categories. A *graph-based approach* pre-processes the agents as a graph, dividing the population into smaller graphs that are each processed by one server. *This is the approach used in this paper*. Examples of such graph partitioning systems include `ParHIP` [Predari et al., 2021] and `Corder` [Chen and Chung, 2021]; see [Schwartz, 2022, Buluç et al., 2016] for comprehensive surveys.

Another approach is a *region-based partitioning*. This applies when the simulation represents a physical space, which can be decomposed into subspaces from top-down approaches (e.g., QuadTree in 2D or KD-Tree in 3D) or bottom-up approaches (e.g., RegionGrowth [Steed and Abou-Haidar, 2003]). This is applicable to situations such as simulating mobility, for instance pedestrian movements in a city and other forms of crowd management [Löhner et al., 2018]. Indeed, pedestrian movements are driven by spatial information (e.g., cannot go through a wall) and agents in the near-vicinity (e.g., to avoid bumping into people). This is not applicable to our work, as our agents do not necessarily map to physical space. Finally, there is a *process-based partitioning* approach, which examines the influence of existing processes onto the workload and then (re)distributes them. For instance, solutions might 'migrate' agents to under-utilized servers, hence several of these solutions are migration-based adaptive heuristic algorithms [Ibrahim et al., 2020].

Since agents are updated according to the same rules, simulations can end up running the same set of instructions over multiple blocks of memory. *General Purpose Graphics Processing Units* (GPGPUs) are often utilized for programs that contain operations that can be performed across multiple units of data. Modern GPUs are many-core processors with hundreds of processing elements, general-purpose instruction sets, and support for double-precision arithmetic [Che et al., 2008]. GPUs have been used in conjunction with traditional processing elements for many applications of ABMs, such as epidemiological simulations [Rao, 2014]. In practice, GPUs tend to accelerate data parallel tasks by about two orders of magnitude when compared to multiple core CPUs [Xiao et al., 2019]. This a direct result of the difference of architecture between the CPU and GPU.

Nvidia's *Compute Unified Device Architecture* (previous designation), or CUDA, is an adapter that gives users the ability to utilize the Nvidia GPUs' instruction set and launch compute kernels. Three technical aspects are important when using CUDA. First, the *communication overhead* should be carefully watched and minimized, otherwise the associated overhead of transferring data (over interconnect) may outweigh the benefits of parallelism. Hence local computations are preferred to communication. Second, unnecessary and expensive method calls should be minimized to reduce kernel context switching, which flushes cached data. Third, the number of threads per block should be carefully tuned to minimize cache misses because one block requesting too many resources can decrease the number of blocks concurrently supported by the GPU [Kosiachenko et al., 2019]. Many past frameworks have used CUDA as they would any ordinary parallel processing framework, hence violating these three

points [Kosiachenko et al., 2019]. Eliminating these inefficiencies has improved performances in ABM simulations.

2.2. Hybrid ABM/FCM models

An FCM serves to transfer the 'mental model' of a real-world individual into the decision-making module of a simulated agent. The knowledge elicitation process to obtain an FCM from a participant has been applied over several decades [Felix et al., 2017] and usually consists of a facilitation session to identify relevant weights and interrelationships, ending with an assessment of causal weights. An FCM represents this knowledge in the form of a labeled, directed, weighted graph (Figure 1). Edge weights encode causation by using a positive weight when an increase in a factor causes an increase in another (e.g., $food\ intake \xrightarrow{+0.24} obesity$) and a negative weight when an increase in a factor leads to a decrease in another (e.g., $stress \xrightarrow{-0.32} food\ intake$). Most importantly, an *FCM is a simulation model*: it synchronously updates the values of the nodes over discrete steps ($t = 0$ and $t = 1$ are shown in Figure 1) by taking into account the current value of each node, the value of its neighbors, and the strengths of their connections. The update process ends when values for a subset of nodes (i.e., the output nodes for which the FCM was created) have stabilized. Similarly to the mental model held internally by an individual, an FCM serves to examine how a person would reflect on a given situation and ultimately arrive at a conclusion [Mkhitaryan and Giabbanelli, 2021].

In an ABM/FCM model, the FCM serves as the 'virtual brain' of each agent. Interactions between agents do not directly happen between their brains, hence only *parts* of an agent's FCM may be externalized via its actions and thus observed by others, leading to changes in their own FCMs. A hybrid ABM/FCM is thus composed of two networks (Figure 2): a social network (each agent is a node, each interaction is an edge) at the population level, and an FCM (each concept is a node, edges denote causality) within each agent. Each simulation step proceeds in *two phases*. First, agents interact via their social ties; for instance, one agent may be told about healthy eating habits by a peer. This information is registered by their FCM as an initial change in the level of 'awareness of healthy practices'. In the next stage, each agent independently reflects on their interactions by simulating their FCM until stabilization. Other factors in the agent's FCM act as a mediating context, hence an initial impetus (e.g., awareness of healthy eating habits) may lead to a change that gets amplified, or may be counter-acted by the agent's context. For instance, an agent may be told that running is healthy, but the agent's elevated level of obesity prevents running (e.g., due to knee pain).

Since their recent introduction [Giabbanelli et al., 2017], hybrid ABM/FCM models have been employed in dozens of studies [Davis et al., 2019]. These studies often focus on socio-environmental problems and involve interdisciplinary teams, hence each subject-matter expert can contribute an FCM and examine how agents equipped with this FCM behave in the simulation. Since FCMs have been used in over 20,000 scientific papers [Kininmonth et al., 2021], the abundance of existing FCMs supports modelers who are interested in creating ABMs where the behavior of each agent is directly informed by a human expert and can be transparently observed. The main drawback has been the limited software for ABM/FCM, as they cannot more than a few thousand agents, which is far from the large populations often used in ABMs to study emerging phenomena.

3. Methods

3.1. Overall Structure

Our new Python library, `cuda-hybrid`, easily allows users to simulate an ABM/FCM hybrid model. The model can be simulated in serial (as is the case with all prior work) or switched to parallel, in which case we automatically leverage the capacities of the user's GPU to accelerate the computations. Note that offering *both* serial and parallel implementations within a single library is necessary to perform benchmarks (Section 4), that is, evaluate the effect of the parallel acceleration. To run an ABM/FCM simulation, the user specifies: the number of agents, the social network of their interactions (e.g., by using a network generator), the FCMs to use (given as files containing edge lists) and the set of concept nodes that must stabilize as well as a stabilization threshold, the number of simulation steps, and the number of repeats. As detailed in the next section, users can also choose which community algorithm they wish to use when executing their model on the GPU, in case they have specific insight into the structure of the social network.

3.2. Running FCMs in parallel via CUDA

Object oriented programming has long been a common approach to build ABMs [Gilbert and Terna, 2000] and it remains the most prominent

paradigm [Cardoso and Ferrando, 2021]. However, the situation is different when building a framework for simulation acceleration. In particular, classes cannot be used within a CUDA-compiled function, and key libraries used in accelerating Python code [Li et al., 2021] (e.g., `numba`) impose a further limitation as they only compile 'a restricted subset of Python code into CUDA kernels'[1]. As a result, the internal routines of our library are not built with object but rather rely on *matrices*, supported by the `numpy` library. Unlike classes, dictionaries, and other Python features, numba supports sending numpy matrices to the GPU. An overall adjacency matrix represents the social network of agents, and each agent has an adjacency matrix for the edge weights of its FCM.

An FCM often consists of only a few dozen concepts, hence its adjacency matrix easily fits within the footprint of CUDA cores. In contrast, the adjacency matrix of a very large number of interacting agents would exceed the memory storage available to many GPUs. It is thus necessary to decompose the overall social network into smaller subparts, similarly to the notion of partitioning discussed in section 2.1. However, the matter is not as trivial as (i) arbitrarily slicing the network into parts and (ii) running computations independently on each one. Indeed, section 2.1 emphasized that the network should be decomposed to ensure a similar workload, so it would be undesirable to have massive subparts and small ones. Additionally, these parts are *not* strictly independent: some agents will necessarily have peers in other partitions so we need to accurately compute the influences that they receive across these partitions (Figure 3). We thus (i) use community detection algorithms to decompose the network and (ii) proceed in two substeps by handling each community in parallel and then reconciling cases with ties across communities. Step (ii) is akin to the parallel processing approach of handling boundary nodes by assigning them 'ghost neighbors' [Riley et al., 2004].

As noted in Section 2.1, one of the important technical aspects when using CUDA is to carefully tune the number of threads per block. In our library, these numbers are automatically selected based on the size of the data needing processed as well as the limitations of the GPU being used.

3.3. User workflow

All of the mechanisms to accelerate the simulation (Section 3.2) happen automatically for the users. Users

[1]See *Overview* at `http://numba.pydata.org/numba-doc/latest/cuda/overview.html`

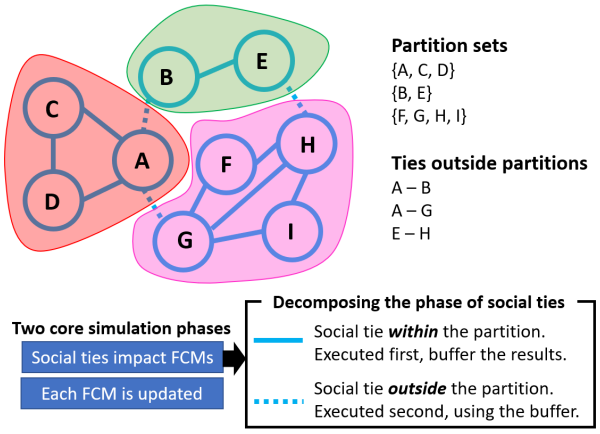

Figure 3: Five agents (A, G, B, E, H) have social ties outside their partition. After all ties within the partitions have been executed, we finish updating these two nodes based on ties outside the partition.

can thus focus on their expertise, by providing the specification for the model that they have designed. To start using our library, a user would provide an FCM as a file; samples for such files are provided in our tutorials at `https://cuda-hybrid.github.io/`. The library reads in the concept nodes, directed edges, edge weights, and maximum number of iterations for the FCM provided. In addition, we allow the user to specify a target concept in the FCM, so that the update stops if this concept stabilizes (Section 2.2). The user would create a social network, typically by calling upon a generator from the `NetworkX` library; our next section experimentally shows the impact of the specific generator used and the targeted network properties. The user has the possibility of initializing the attributes of each agents, that is, the node values for their FCM. These values are randomized by default.

Most of the code written by the user serves to specify how two FCMs influence one another via a social tie (pink dotted line in Figure 2). Consider two agents A and B, where A influences B. In line with previous software [Giabbanelli et al., 2019], the user designates a subset of FCM factors in B that are *influenced*, and a (possibly distinct) subset of factors in A that are *influencing*. Then, the user writes the exact nature of this influence. For example, consider that A has diabetes type-II and B does not. B cannot directly get diabetes from A, as it is not contagious. However, B's now has a heightened awareness of diabetes. The user may program that B's awareness of diabetes goes up exponentially for each peer with diabetes. This does not need to be programmed for every single pair of agent; it is entered only once and applied for every interaction.

The example above shows a *relative* influence, as the next level of B's awareness increases from its current level. FCM nodes modified by relative influences can accumulate several such sources of influence; for instance, B's awareness would go up even more if a peer got amputated for diabetes, hence several influencing factors (diabetes, amputation) can be applied jointly. Users can also specify an *absolute* influence, whereby the value of a concept is replaced irrespectively of its previous level. For instance, in a simplified model of obesity, if most of B's peers are obese then B becomes obese [Bahr et al., 2009]. The user *should not* attempt to change a factor with both relative and absolute influences (e.g., the value should go up by 5% and at the same time by replaced by 0.25), because the *order* in which these influences are executed would matter and hence violate the principle of independence that underpins parallelism. However, the library is not responsible for a user's erroneous model specifications.

4. Case Study on Nutrition

The objectives of our case study are twofold: (i) demonstrate that the library is correctly implemented, by its ability at replicating results obtained in previous (serial) software; and (ii) demonstrate that the library does lead to scalability, by running a much larger population of agents than previously possible by the serial approach. The latter is achieved through the workflow in Figure 4. The FCM used comes from a case study related to obesity in Canada [Giabbanelli et al., 2014], and it focuses on the social transmission of knowledge regarding nutrition (Figure 5). To closely replicate the previous study, we followed its process to set the initial values for FCM concepts in the agents (e.g., stratified per age and income, generated from inverse Gaussian distributions) and to specify the influencing equation (section 3.3). The level of nutrition knowledge of an influenced agent j depends on the knowledge of the influencing agent i in a probabilistic manner, whereby there is a chance p of accepting a good advice (increasing one's knowledge if the peer knows more) and a chance $1 - p$ of accepting a bad advice (decreasing one's knowledge by following the sub-par practices of a peer).

Since scalability may be affected by the algorithm chose to subdivide the social network and/or the generators used to create the network, we used different methods for these two aspects. *Two different community algorithms* were used to examine the effect of subdividing the network. *Clauset-Newman-Moore greedy modularity maximization* started with each node in a separate community, and then combined communities that increased modularity the most. Modularity is defined as a network property that describes the tendency of the nodes to cluster [Gilarranz et al., 2017] [Newman, 2019, p. 498]. The second algorithm, *label propagation*, created communities by combining synchronous and asynchronous models while simultaneously using a semi-synchronous label propagation method. We ensured that these algorithms did not group the nodes into one entire community as other well-known community algorithms (e.g. k_clique_communities).

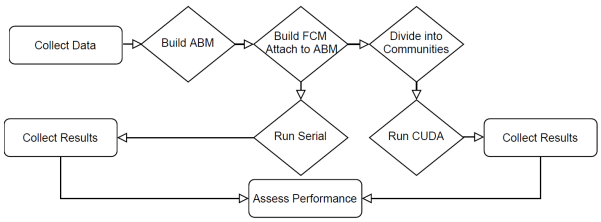

Figure 4: Methods Overview.

Table 2: Social networks of 60,000 agents obtained by fine-tuning the parameters of four generators: *scale_free_graph* (1 parameter), *barabasi_albert_graph* (2 parameters), *watts_strogatz_graph* and *newman_watts_strogatz_graph* (3 parameters each).

Edge Count per Graph Generator	
Graph Generator	Number of Edges
barabasi_albert_graph	119996
scale_free_graph	130690
newman_watts_strogatz_graph	120000
watts_strogatz_graph	120000

For the network structure, we considered two commonly encountered types of social networks (scale-free and small-world; see section 2.1) and used two generators for each one via `NetworkX`: *watts_strogatz_graph* and *newman_watts_strogatz_graph* for small-world networks, *scale_free_graph* and *barabasi_albert_graph* for scale-free networks. In addition, we fine-tuned the generators' parameters to create a similar number of edges for each type of network. This fine-tuning is necessary, otherwise results would be reflecting the different densities of the network rather than the ways in which the social ties are positioned. Results from the tine-tuning are exemplified for 60,000 agents in Table 2. Note that the parameters of the network generators impose certain constraints [Freund and Giabbanelli, 2022], hence it is not always possible to obtain networks that have *exactly* the same number of edges (Table 2).

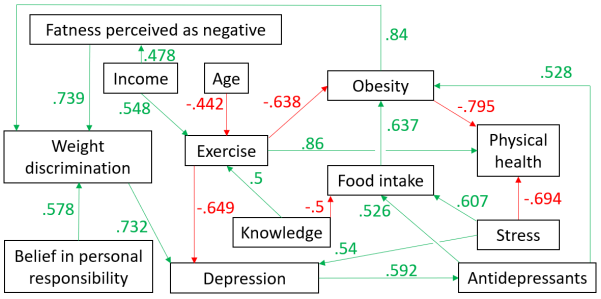

Figure 5: Obesity Fuzzy Cognitive Map.

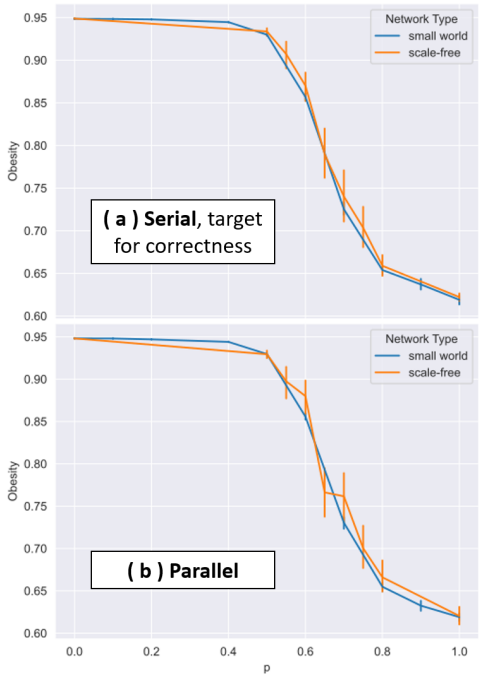

Figure 6: Average level of obesity in the population (y-axis) in response to the probability of influence p in both a traditional serial execution of the model (a; top) and via our parallel approach (b; bottom).

5. Results

5.1. Experimental Setup and Correctness

To *replicate* the results of the previous study running on a serial platform, we ensured that the networks were as similar as possible to that study. Results show that the trend from a serial execution (Figure 6-a) resemble the trends obtained in our parallel execution (Figure 6-b). We observe discrepancies for the scale-free graph for parameter values p between 0.6 and 0.75, but we note that there are multiple sources of randomness (initialization of the agents' attributes, influence probability p, network generator) hence the difference can be attributed to the sensitivity of scale-free graphs to certain probabilistic events (e.g., if a hub in the graph adopts a certain behavior then it has a disproportionate impact on the rest of the dynamics).

5.2. Scalability

To *assess scalability*, we computed the results on a serial implementation with all four network generators, and on a parallel implementation with all four network generators as well as both community detection algorithms (8 combinations). For a fair comparison, we did not artificially burden the serial implementation by performing a community division, since this additional processing is only needed for parallelism. The parallel implementation used a GPU with 5,120 cores (NVIDIA Tesla V100 PCIe) and 16 Gb GB HBM2 memory. We ran from $1,000$ up to $60,000$ agents and each simulation was repeated 10 times. The difference in execution time (Figure 7) shows that a parallel execution runs much faster. On our hardware, the serial implementation is limited to $5,000$ agents whereas the parallel case handles up to $108,000$ agents. To better appreciate this difference and bring nuance into the occasionally overlapping visuals in close cases, Table 3 lists the average runtime for the same population size $(1,000)$ and summarizes the parallel speed-up, which ranges from 26x to 126x depending on the network structure and generator.

While the visible effect of network structure and generator on serial processing time *appears* to be inconsequential for parallel processing (from 0.149 to 0.191 minute), that is only because a population of $1,000$ agents is processed very quickly in parallel. At larger population sizes, we observed that the structure and generator also matter in parallel. In addition, the effect of community detection algorithms also becomes noticeable. For example, at $60,000$ agents, the scale-free generator followed by the greedy community detection completes in 48.064 minutes, but with the label propagation it takes only half as long (24.609 minutes). One community detection is not *always* faster than the other: a small-world network via Watts-Strogatz takes 15.769 minutes with the greedy approach, but in this situation label propagation is longer (21.780 minutes).

6. Discussion

Although ABM/FCM hybrid models are an increasingly popular modeling approach for self-adaptive systems, software technologies were limited to small population sizes that did not suffice

Table 3: Differences in runtime (minute) between the serial and parallel executions. The speed-up is computed as the average between the two different community algorithms that may be used for a parallel implementation.

Network structure	Generator	Community (parallel only)	Serial time	Parallel time	Parallel speed-up
scale-free	scale-free	Greedy	4.236	0.165	26x
		Label		0.159	
	Barabasi	Greedy	4.394	0.191	26x
		Label		0.150	
small-world	Newman	Greedy	11.712	0.149	74x
		Label		0.165	
	Watts	Greedy	19.354	0.152	126x
		Label		0.154	

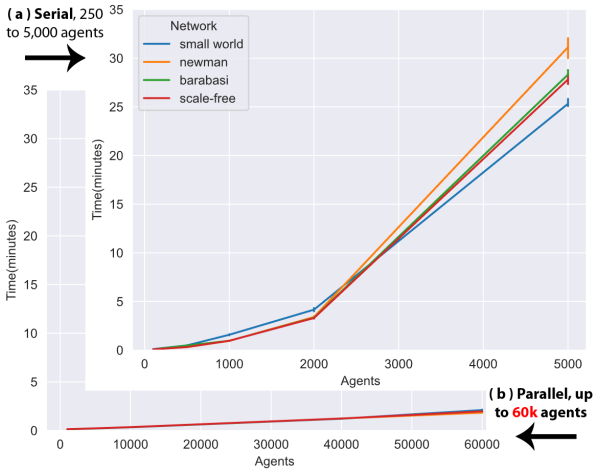

Figure 7: Compute time (y-axis) depending on the number of agents for traditional serial execution of the model (a) and via our parallel approach (b). Note that the two x-axes have different scales, since a serial execution stops handling ABMs at 5,000 agents.

to study emerging phenomena. In this paper, we developed a library that automatically leverages CUDA cores available in GPUs to accelerate the simulation. Our results confirm the correctness of our implementation and the presence of a massive parallel speed-up, from 26x to 126x at small population sizes. For larger population sizes, there is no notion of speedup because only the parallel implementation could continue to perform simulations. Although we expect the runtime to depend on the structure of the population (e.g., small-world networks have dense communities), we found that the choice of network generator had an impact, as well as the choice of the community detection algorithm used to divide the population (such that it fits in memory). It is likely that the rules of a model also impact runtime, which could be assessed by future work comparing ABM/FCM of different designs, such as the model used here (for nutrition in human) with a model of interactions between fish in [Wang et al., 2006].

Even though we made great use of parallelism through CUDA, there was relatively little use of CPU, or host parallelism in our work. This application could be greatly improved by multi-threading the code that occurs on the host: network generation, community detection, and running model replication. Multithreading could be done using the Python `multiprocessing library`, which would work better with NetworkX. Future work may also explore other libraries to access the GPU, such as the backends provided by PyTorch.

One limitation of our library is that it does not run on all GPUs. Because we chose to use the dominant player on the market to develop this application (hence making it more likely that our users can leverage the library for their hardware), there is a degree of vendor lock-in. Rather than writing a version of the library for every manufacturer, future works could explore emerging solutions that can be deployed across accelerators (e.g., various GPUs, FPGA), such as Intel oneAPI [Nozal and Bosque, 2021]. A second limitation resulting from the hardware is that our number of nodes eventually exceeded the memory available on the GPU (Nvidia Tesla). One contributing factor to our memory shortage was the representation we used for each graph in the application. We preferred an adjacency matrix over an adjacency list. This increased both processing speed and memory consumption. However, even with improvements in data structures, some scientific applications require a very large memory capacity and do not fit into a GPU [Emani et al., 2021]. In such cases, another architecture is necessary, and the switch could be made seamlessly when using a unified application programming interface such as Intel oneAPI.

Another potential improvement to our work pertains

to its limited capacity to visualize the simulation as it occurs. Although most users may only be interested in the end results of a simulation, subject matter experts can sometimes identify potential mistakes in a simulation by looking at a visualization of steps. In addition, intermediate results may show sufficiently clear trends that we can confidently predict the conclusion of a simulation without needing to perform all remaining steps [Lutz and Giabbanelli, 2022]. In both cases, stopping a simulation earlier could be time-saving, particularly for large populations. Since we operate in Python, visualization of unfolding simulation results could be performed within a Jupyter notebook that orchestrates the deployment and analysis of results [Savira et al., 2021].

7. Conclusion

We created the first ABM/FCM library that runs in parallel, which opens the possibility to simulate agent populations in the dozens of thousands within less than an hour instead of being limited to about 5,000 agents with prior implementations. We revealed that factors occasionally ignored (e.g., the choice of network generator) can ultimately impact compute time, but the optimal combination of factors remains elusive as the effect of several parameters (e.g., community detection algorithm, features of the ABM/FCM model) would require a much larger comparative study.

References

[Abar et al., 2017] Abar, S., Theodoropoulos, G. K., Lemarinier, P., and O'Hare, G. M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33.

[Abdou et al., 2012] Abdou, M., Hamill, L., and Gilbert, N. (2012). Designing and building an agent-based model. In *Agent-based models of geographical systems*, pages 141–165. Springer.

[Amblard et al., 2015] Amblard, F., Bouadjio-Boulic, A., Gutiérrez, C. S., and Gaudou, B. (2015). Which models are used in social simulation to generate social networks? a review of 17 years of publications in jasss. In *Winter Simulation Conference (WSC)*, pages 4021–4032. IEEE.

[Bahr et al., 2009] Bahr, D. B., Browning, R. C., Wyatt, H. R., and Hill, J. O. (2009). Exploiting social networks to mitigate the obesity epidemic. *Obesity*, 17(4):723–728.

[Barde and Van Der Hoog, 2017] Barde, S. and Van Der Hoog, S. (2017). An empirical validation protocol for large-scale agent-based models.

[Buluç et al., 2016] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. (2016). Recent advances in graph partitioning. *Algorithm engineering*, pages 117–158.

[Cambier et al., 2002] Cambier, C., Piron, M., and Cardon, A. (2002). Self-adaptive systems using a massive multi-agent system. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 345–350. IEEE.

[Cardoso and Ferrando, 2021] Cardoso, R. C. and Ferrando, A. (2021). A review of agent-based programming for multi-agent systems. *Computers*, 10(2):16.

[Che et al., 2008] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., and Skadron, K. (2008). A performance study of general-purpose applications on graphics processors using cuda. *Journal of Parallel and Distributed Computing*, 68(10):1370–1380. General-Purpose Processing using Graphics Processing Units.

[Chen and Chung, 2021] Chen, Y. and Chung, Y.-C. (2021). Workload balancing via graph reordering on multicore systems. *IEEE Transactions on Parallel and Distributed Systems*, 33(5):1231–1245.

[Davis et al., 2019] Davis, C. W., Giabbanelli, P. J., and Jetter, A. J. (2019). The intersection of agent based models and fuzzy cognitive maps: a review of an emerging hybrid modeling practice. In *Winter Simulation Conference (WSC)*, pages 1292–1303. IEEE.

[Emani et al., 2021] Emani, M., Vishwanath, V., Adams, C., Papka, M. E., Stevens, R., Florescu, L., Jairath, S., Liu, W., Nama, T., and Sujeeth, A. (2021). Accelerating scientific applications with sambanova reconfigurable dataflow architecture. *Computing in Science & Engineering*, 23(2):114–119.

[Felix et al., 2017] Felix, G., Nápoles, G., Falcon, R., Froelich, W., Vanhoof, K., and Bello, R. (2017). A review on methods and software for fuzzy cognitive maps. *Artificial Intelligence Review*, 52(3):1707–1737.

[Freund and Giabbanelli, 2021] Freund, A. J. and Giabbanelli, P. J. (2021). The necessity and difficulty of navigating uncertainty to develop an individual-level computational model. In *International Conference on Computational Science*, pages 407–421. Springer.

[Freund and Giabbanelli, 2022] Freund, A. J. and Giabbanelli, P. J. (2022). An experimental study on the scalability of recent node centrality metrics in sparse complex networks. *Frontiers in big Data*, 5.

[Giabbanelli et al., 2019] Giabbanelli, P., Fattoruso, M., and Norman, M. L. (2019). Cofluences: Simulating the spread of social influences via a hybrid agent-based/fuzzy cognitive maps architecture. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '19, page 71–82, New York, NY, USA. Association for Computing Machinery.

[Giabbanelli et al., 2017] Giabbanelli, P. J., Gray, S. A., and Aminpour, P. (2017). Combining fuzzy cognitive maps with agent-based modeling: Frameworks and pitfalls of a powerful hybrid modeling approach to understand human-environment interactions. *Environmental Modelling & Software*, 95:320–325.

[Giabbanelli et al., 2014] Giabbanelli, P. J., Jackson, P. J., and Finegood, D. T. (2014). Modelling the joint effect of social determinants and peers on obesity among canadian adults. In *Theories and simulations of complex social systems*, pages 145–160. Springer.

[Gilarranz et al., 2017] Gilarranz, L. J., Rayfield, B., Liñán-Cembrano, G., Bascompte, J., and Gonzalez, A. (2017). Effects of network modularity on the spread of perturbation impact in experimental metapopulations. *Science*, 357(6347):199–201.

[Gilbert and Terna, 2000] Gilbert, N. and Terna, P. (2000). How to build and use agent-based models in social science. *Mind & Society*, 1(1):57–72.

[Grantham and Giabbanelli, 2020] Grantham, E. O. and Giabbanelli, P. J. (2020). Creating perceptual uncertainty in agent-based models with social interactions. In *2020 Spring Simulation Conference (SpringSim)*, pages 1–12.

[Hesam et al., 2021] Hesam, A., Breitwieser, L., Rademakers, F., and Al-Ars, Z. (2021). Gpu acceleration of 3d agent-based biological simulations. *arXiv preprint arXiv:2105.00039*.

[Ibrahim et al., 2020] Ibrahim, M., Iqbal, M. A., Aleem, M., Islam, M. A., and Vo, N.-S. (2020). Maha: Migration-based adaptive heuristic algorithm for large-scale network simulations. *Cluster Computing*, 23(2):1251–1266.

[Kininmonth et al., 2021] Kininmonth, S., Gray, S., and Kok, K. (2021). *Expert modelling*, page 231. Taylor and Francis, 711 3rd Ave New York, NY.

[Kosiachenko et al., 2019] Kosiachenko, L., Hart, N., and Fukuda, M. (2019). Mass cuda: A general gpu parallelization framework for agent-based models. In Demazeau, Y., Matson, E., Corchado, J. M., and De la Prieta, F., editors, *Advances in Practical Applications of Survivable Agents and Multi-Agent Systems: The PAAMS Collection*, pages 139–152, Cham. Springer.

[Krupitzer et al., 2015] Krupitzer, C., Roth, F. M., VanSyckel, S., Schiele, G., and Becker, C. (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206.

[Lavin and Giabbanelli, 2017] Lavin, E. A. and Giabbanelli, P. J. (2017). Analyzing and simplifying model uncertainty in fuzzy cognitive maps. In *Winter Simulation Conference (WSC)*, pages 1868–1879. IEEE.

[Li and Giabbanelli, 2021] Li, J. and Giabbanelli, P. J. (2021). Returning to a normal life via covid-19 vaccines in the usa: a large-scale agent-based simulation study. *medRxiv*.

[Li et al., 2021] Li, J., Giabbanelli, P. J., and Köster, T. (2021). Comparing the effect of code optimizations on simulation runtime across synchronous cellular automata models of hiv. In *Winter Simulation Conference (WSC)*, pages 1–12. IEEE.

[Löhner et al., 2018] Löhner, R., Haug, E., Zinggerling, C., and Oñate, E. (2018). Real-time micro-modelling of city evacuations. *Computational Particle Mechanics*, 5(1):71–86.

[Lui and Chan, 2002] Lui, J. C. S. and Chan, M. (2002). An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transactions on parallel and distributed systems*, 13(3):193–211.

[Lutz and Giabbanelli, 2022] Lutz, C. B. and Giabbanelli, P. J. (2022). When do we need massive computations to perform detailed covid-19 simulations? *Advanced Theory and Simulations*, 5(2):2100343.

[Mehryar et al., 2019] Mehryar, S., Sliuzas, R., Schwarz, N., Sharifi, A., and van Maarseveen, M. (2019). From individual fuzzy cognitive maps to agent based models: Modeling multi-factorial and multi-stakeholder decision-making for water scarcity. *Journal of environmental management*, 250:109482.

[Mkhitaryan and Giabbanelli, 2021] Mkhitaryan, S. and Giabbanelli, P. J. (2021). How modeling methods for fuzzy cognitive mapping can benefit from psychology research. In *Winter Simulation Conference (WSC)*, pages 1–12. IEEE.

[Newman, 2019] Newman, M. E. (2019). *Networks*. Oxford University Press.

[Nozal and Bosque, 2021] Nozal, R. and Bosque, J. L. (2021). Exploiting co-execution with oneapi: heterogeneity from a modern perspective. In *European Conference on Parallel Processing*, pages 501–516. Springer.

[Predari et al., 2021] Predari, M., Tzovas, C., Schulz, C., and Meyerhenke, H. (2021). An mpi-based algorithm for mapping complex networks onto hierarchical architectures. In *European Conference on Parallel Processing*, pages 167–182. Springer.

[Qureshi and Perini, 2008] Qureshi, N. A. and Perini, A. (2008). An agent-based middleware for adaptive systems. In *2008 The Eighth International Conference on Quality Software*, pages 423–428. IEEE.

[Rao, 2014] Rao, D. M. (2014). Accelerating parallel agent-based epidemiological simulations. In *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '14, page 127–138, New York, NY, USA. Association for Computing Machinery.

[Richmond et al., 2010] Richmond, P., Walker, D., Coakley, S., and Romano, D. (2010). High performance cellular level agent-based simulation with flame for the gpu. *Briefings in bioinformatics*, 11(3):334–347.

[Riley et al., 2004] Riley, G. F., Jaafar, T. M., Fujimoto, R. M., and Ammar, M. H. (2004). Space-parallel network simulations using ghosts. In *18th Workshop on Parallel and Distributed Simulation (PADS)*, pages 170–177. IEEE.

[Savira et al., 2021] Savira, P., Marrinan, T., and Papka, M. E. (2021). Writing, running, and analyzing large-scale scientific simulations with jupyter notebooks. In *2021 IEEE 11th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 90–91. IEEE.

[Schwartz, 2022] Schwartz, S. (2022). An overview of graph covering and partitioning. *Discrete Mathematics*, 345(8):112884.

[Steed and Abou-Haidar, 2003] Steed, A. and Abou-Haidar, R. (2003). Partitioning crowded virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 7–14.

[Stula et al., 2010] Stula, M., Stipanicev, D., and Bodrozic, L. (2010). Intelligent modeling with agent-based fuzzy cognitive map. *International journal of intelligent systems*, 25(10):981–1004.

[Tang and Bennett, 2011] Tang, W. and Bennett, D. A. (2011). Parallel agent-based modeling of spatial opinion diffusion accelerated using graphics processing units. *Ecological Modelling*, 222(19):3605–3615.

[Taylor, 2019] Taylor, S. J. (2019). Distributed simulation: State-of-the-art and potential for operational research. *European Journal of Operational Research*, 273(1):1–19.

[Voinov et al., 2018] Voinov, A., Jenni, K., Gray, S., Kolagani, N., Glynn, P. D., Bommel, P., Prell, C., Zellner, M., Paolisso, M., Jordan, R., et al. (2018). Tools and methods in participatory modeling: Selecting the right tool for the job. *Environmental Modelling & Software*, 109:232–255.

[Wang et al., 2006] Wang, D., Berry, M. W., Carr, E. A., and Gross, L. J. (2006). A parallel fish landscape model for ecosystem modeling. *SIMULATION*, 82(7):451–465.

[Xiao et al., 2019] Xiao, J., Andelfinger, P., Eckhoff, D., Cai, W., and Knoll, A. (2019). A survey on agent-based simulation using hardware accelerators. *ACM Comput. Surv.*, 51(6).