# Large-scale parallelization of Human Migration Simulation

Derek Groen, Nikela Papadopoulou, Petros Anastasiadis, Marcin Lawenda, Lukasz Szustak, Sergiy Gogolenko, Hamid Arabnejad, Alireza Jahani

**Abstract**—Forced displacement of people worldwide, for example due to violent conflicts, is common in the modern world, and today more than 82 million people are forcibly displaced. This puts the problem of migration at the forefront of the most important problems of humanity. The Flee simulation code is an agent-based modelling tool that can forecast population displacements in civil war settings, but performing accurate simulations requires non-negligible computational capacity. In this paper we present our approach to Flee parallelization for fast execution on multi-core platforms, as well as discuss the computational complexity of the algorithm and its implementation. We benchmark parallelized code using a supercomputers equipped with AMD EPYC Rome 7742 and Intel Xeon Platinum 8268 processors and investigate its performance across a range of alternative rule sets, different refinements in the spatial representation and various numbers of agents representing displaced persons. We find that Flee scales excellently to up to 8,192 cores for large cases, although very detailed location graphs can impose a large initialization time overhead.

**Index Terms**—Migration, Refugees, Global Systems Science, Global Challenges, parallelization, HPC, AMD Rome, Intel Xeon, benchmarks, modeling, computational complexity.

## 1 Introduction

In today's world, the issue of human migration is of huge importance to the global community, with over 82 million people forcible displaced [48] and immigration policies being one of the major topics in the media. Forecasting the movements and destinations of people displaced by conflict is useful, as it can guide the implementation and management of humanitarian support efforts. One way to provide these forecasts is through simulation. In this paper, we present a parallel implementation of the Flee simulation code and analyze its performance under a wide range of conditions. Flee forecasts the destinations of people escaping violent conflicts, and has been previously validated for a range of geographically different conflicts [45], [46], [29], [52] using the FabSim3 automation toolkit [25].

Social simulation is a computational method that aims to study issues in the social sciences including problems in psychology, organizational behavior, sociology, political science, economics, anthropology, geography, and humanitarian research [21]. It combines the descriptive abstraction of a social system with a process-centric or behavior-centric algorithm to reconstruct the social reality in a virtual environment, and execute the human activities to support reasoning in decision making. This field explores the simulation of societies as complex adaptive non-linear systems, which are difficult to study with classical mathematical equation-based models. And in which a perfect understanding of the individual parts does not automatically convey a perfect understanding of the whole system's behavior, especially when there is only sparse data available and only partial knowledge of the real-world phenomenon is at hand [34]. Agent-based social simulation (ABSS) is a variant of computational social systems [14], that involves modeling different virtual societies in which a population of independent agents, varying on time and spatial scales, with individual behaviors defined through a set of rules, interact with each other across a logical network. The main goal of the simulated society is to observe the behaviors of the agents and synthesize simulated data to learn about the reactions of the artificial agents and their combined effects evolved. This helps in studying the descriptive and perspective view of the social system in question and building an inferential reasoning framework that assists in real-world decision-making.

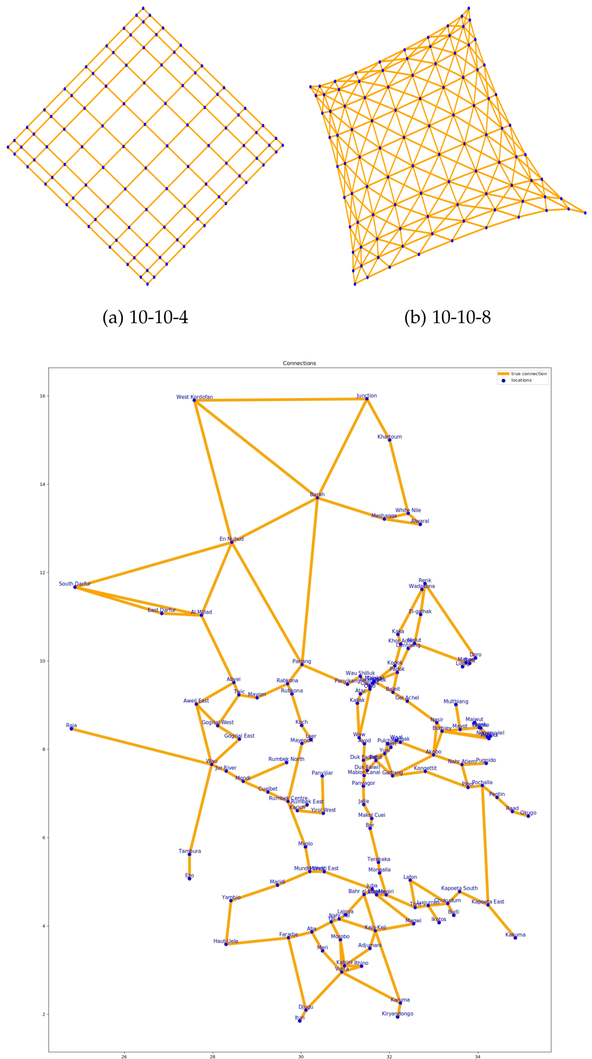
In this work, we use agent-based social simulations to model and analyze a crucial problem in humanitarian research: forced human migration. Forced human migration is the involuntary relocation of people away from their homelands due to a variety of factors including natural disasters, violence, ethnic cleansing, individual or group

*Derek Groen, Hamid Arabnejad and Alireza Jahani are with the Department of Computer Science, Brunel University London, Uxbridge, UB8 3PH, UK.*
*Nikela Papadopoulou is with the Department of Computer Science and Engineering, Chalmers University of Technology, Sweden, and with the Computing Systems Laboratory, National Technical University of Athens, Greece.*
*Petros Anastasiadis is with the Computing Systems Laboratory, National Technical University of Athens, Greece.*
*Marcin Lawenda is with the Poznan Supercomputing and Networking Center, Poznan, Poland.*
*Lukasz Szustak is with the Poznan Supercomputing and Networking Center, Poznan, Poland as well as Czestochowa University of Technology, Czestochowa, Poland.*
*Sergiy Gogolenko is with the High Performance Computing Center Stuttgart, Nobelstr. 19, 70569, Stuttgart, Germany.*
*This work has been supported by the HiDALGO project and has been partly funded by the European Commission's ICT activity of the H2020 Programme under grant agreement number: 824115. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.*

persecution, droughts, civil wars, deportation, and/or other religious and social reasons [18]. According to the UN Refugee Agency (UNHCR) more than 70.8 million people are forcibly displaced worldwide. Among them, 25.9 million are refugees, half of which fled from Syria, Afghanistan, and South Sudan [48]. These fleeing individuals are the unfortunate victims of internal armed conflicts and civil wars, who make decisions to migrate in times of distress. Their decisions are often based on economic and political push and pull factors in sending and receiving countries. Researchers have mostly investigated why human migration occurs and its effect on economies using migration theories and econometric models, with little attention paid to predicting forced human migration. In addition, migration data is often missing, incomplete, or only available at long intervals. As a result, existing models largely base on regressing existing forced migration data, limiting their predictive power with incomplete or short datasets.

In this area of research, it is important to be able to predict where refugees go because: (i) It helps governments and NGOs to correctly allocate humanitarian resources to the refugee camps and save refugee lives; (ii) It helps complete incomplete data collections on refugee movements; and (iii) It helps to investigate the consequences of a nation closing its border for refugees. To address the main goals of this social problem, we use an agent-based social simulation development approach (SDA) [45] that allows us to forecast the movements of forcibly displaced people in conflicts. We model refugees as agents with defined behaviors using a set of rules and are placed within a virtual environment defined by a geospatial location graph. The agent model, virtual environment, and the set of rules all together constitute an agent-based social simulation framework: Flee.

The parallel version of Flee, called PFlee, is a highly efficient and scalable code that helps to easily simulate complex simulation scenarios of forced migration occurring in different parts of the world. The main motivation for developing PFlee is firstly to enable social scientists and researchers to perform large-scale simulation runs with (a) a large number of conflict locations, camps, cities, towns, and settlements and (b) a large number of refugees moving across these locations. As highlighted in [11], most existing approaches in parallel ABMS target a parallel parameter sweep and not scale-out parallelization of ABMS, while recent work recognizes the need for high-performance ABMS [42], and another recent approach employs Markov aggregation for more computationally efficient simulation [19]. Secondly, PFlee seeks to enable modelers and developers to extend the framework by focusing on the complexity of the conceptual model with more complex agent behaviors, rule sets and/or virtual environments, without worrying about the time and computational complexity of the framework. PFlee is written in Python, offering programming ease, allowing for high programming and scientific productivity, and fast prototyping. It offers efficient execution on any type of state-of-practice or state-of-the-art cluster of multicore CPUs, making use of open-source Python libraries for parallelization and efficiency. With the development of PFlee, we have achieved reducing simulation time from a few hours to a few minutes or seconds. The Flee (and PFlee) agent-based modeling toolkit is publicly available as open-source software [1], under a BSD-3-Clause license.



(a) 10-10-4 (b) 10-10-8

(c) South Sudan Location Graph

Fig. 1: Graphical overview of 3 location graphs, including 10-10-4 (top left), 10-10-8 (top right) and the South Sudan location graph (bottom). The 50-50-4 and 100-100-4 location graphs are larger versions of 10-10-4 with the same connectivity characteristics. The 50-50-8 and 100-100-8 location graphs are larger versions of 10-10-8 with the same connectivity characteristics.

The remaining of this paper is organized as follows. Section 2 reviews state-of-the-art tools commonly used to implement networked ABMs. Section 3 presents the main algorithms used in Flee to model the movements of agents, sequentially and in parallel. In the same section, we also present our approach to distributing information (locations and agent state updates) across processes, balancing the computational load. Section 4 describes performance optimizations in PFlee using the Numba library in Python. Section 5 evaluates the performance of PFlee using different scenarios on two large-scale systems employing different processor architectures: AMD EPYC Rome and Intel Xeon. Section 6 discusses the scientific results of PFlee simulations

1. https://github.com/djgroen/flee-release

from the perspective of migration research. Finally, Section 7 concludes the paper and describes future work.

## 2 Related work

In this section, we present the wider context in which our parallelization effort of the Flee migration modeling code takes place. We discuss two aspects: human migration modeling and large individual- and network-based social simulations.

### 2.1 Human migration modelling

In the context of human migration, there is a range of work that is relevant to our research. As a foundational contribution, Edwards [15] highlights the potential use of computational models in predicting key spatial patterns of conflict-induced forced displacement. In terms of examples, Hebert et al. [27] propose an agent-based model of Syrian refugees to predict their movements and behavior on destination selection based on their respective characteristics and needs. Sokolowski et al. [44] propose an agent-based model using real-world data of the Syrian cities and the demographics of the city population to simulate Syrian population displacement. Collins at al. [12] introduce a strategic group formation mechanic into an ABM to investigate the impact on refugee evacuation time. Picascia et al. [39] present an agent-based simulation of housing in urban Beirut as a tool for policy-making and what-if questions about the urban environments in the context of migration. Liu at al. [32] develop a simulation of refugee flow in Europe using a multi-objective optimization dynamic programming model to compute the best allocation of resources in different regions. Hattle et al. [26] model the refugee immigration flow of Syrian refugees to and through Europe using an agent-based approach. Other relevant works focus on population simulation, rather than migration, offering approaches that combine agent-based modeling with micro-simulation [53], and deep neural networks to model the decision making [54].

The aforementioned human migration simulations mostly focus on the social aspects of the simulation and lack high performance and large-scale implementations. Only a scarce set of works focus on high-performance and large-scale agent-based simulations, and almost no works address human migration. Blandin et al., [6] propose a parallel *Python* implementation for human migration and scale up to a population of 7 billion agents at a macro scale resolution (global scale), however, the micro-scale agent behaviors have been largely ignored. In this paper, we present a parallel implementation of agent-based human migration simulation with sufficient scalability while keeping an account for the micro-level agent behavior e.g., individual decisions, methods of transport, and awareness levels.

### 2.2 Large individual- and network-based social simulations

Large individual- and network-based social simulations form a fundamental pillar for social simulations in computational global system science (GSS) applications from a wide range of domains such as epidemiology [20], [8], [17], [2], social networks modelling [50], economics and logistics including supply chains [41], as well as urban planning [28], [47] including transportation modelling [36]. Many of these models can be categorized into four groups according to the network structure: hierarchical (tree-like) models, social contact networks, plain and multilayer networks, and interconnected geo-social networks,[22].

Hierarchical models originate from computational epidemiology, where they serve to model physical interactions in society, in a greatly simplified manner, with static multilevel trees [20], [8], [38]. The root of the hierarchical model tree corresponds to the whole society, while the lower levels of the tree represent elements of the society with finer granularity, until reaching the level of individual households as leaves. In particular, the intermediate layers can correspond to countries, states, regions, counties, municipalities, settlements, and city blocks. Agents are assigned to the leaves of the tree. Distance to the closest common parent determines the probability of interactions between two agents. With hierarchical models, the fastest known simulation corresponds to C++ codes for pandemics simulation by K. Perumalla and S. Seal [38] which reports speed up 10’000 on 64K cores (15.2% efficiency) on Cray XT5.

Social contact networks (SCN) were also introduced first in the context of computational epidemiology [16], [17] to enrich the set of policies for analysis and reflect the real-world social networks more accurately compared to the tree-like models. SCN captures agent interactions by a bipartite agent-to-location (A2L) graph $G_{A2L} = (V_{\mathcal{A}}, V_{\mathcal{L}}, E)$ – with agents $\mathcal{A}$ on the one side and loci of their interactions $\mathcal{L}$ on the other – supplemented by the stochastic schedule $W$ for agents to visit loci of interactions. SCNs gained a solid attention in HPC community, where researchers developed a significant number of optimizations to scale SCNs on distributed HPC environments [2], [55], [4], [50]. These studies evaluate impact of completion detection synchronization [55], message aggregation [55], as well as different load balancing strategies including round-robin data distribution [2], partitioning focusing on edge cuts [55], and geographic partitioning [4]. In order to address peculiarities of real-world data related to skewed and heavy-tailed degree distributions, the authors analyse performance of hub location [55], [50] and agent [50] decomposition. In [4], the authors reported speed up of 182’073 (23% efficiency) on 786K cores of Blue Gene/Q with the Charm++ codes for pandemics simulation called EpiSimdemics.

Plain and multilayer networks usually emerge in two types of large-scale agent-based models. In the first type, the network represents the environment, often static, in which agents are assigned to the vertices of the network and can relocate only to the neighboring vertices in each step. Such models are common in transportation modelling [36]. In the second type, the plain network corresponds to the evolving agent-to-agent (A2A) network which either stems from reduction of SCN [5], [47] or represents aspatial (“soup”) model of social connections per-se.

Interconnected geo-social networks (IGSN) combine idea of SCN and multilayer networks [28]. IGSN $G = (G_S, G_L, I)$ consists of two graph layers – geographic $G_L = (V_{\mathcal{L}}, E_{\mathcal{L}})$ and social $G_S = (V_{\mathcal{A}}, E_{\mathcal{A}})$ – with the interlayer edge set $I$. This group of networked agent-based models is less studied compared to the former three.

During the last decade, researchers developed a vast number of HPC compliant codes to support the implementation of agent-based models, a thorough review of which lies beyond the scope of this paper. Instead, we briefly review state-of-the-art tools commonly used to implement networked ABMs. For an overview of parallel and distributed agent-based systems (PDABS), we refer the interested readers to [30], [40], [1]. In [51], authors survey ABMS using hardware accelerators. RepastHPC and D-MASON constitute the two most popular general-purpose PDABS suitable for networked agent-based simulations. Written in C++03 with MPI-based communication layer, RepastHPC has formally all components required to build networked ABMs. An example of large population simulation with RepastHPC is presented in [3]. Nevertheless, the latest version of RepastHPC neglects some optimizations such as agent decomposition [50] and recent advances in high-performance data structures (e.g., new techniques for handling evolving graphs and modern implementations of hash tables). In [9], [11], the authors report the scalability of RepastHPC on basic networked models to 32K cores. D-MASON is a distributed version of Java-based MASON framework with MPI communication layer [13] which scales to hundreds of logical processes [13], [49]. It suites for implementation of ABMs with plain and multi-layer networks and offers out-of-the-box integration with GIS. In many occasions, ABMs with plain networks can be efficiently implemented on top of highly optimized distributed graph-parallel frameworks like PowerGraph [23], GraphX [24], GraphChi [31], and Ligra [43].

## 3 Methodology

The Flee agent-based modeling code calculates the daily movement of displaced agents. Agents are explicitly resolved as Python objects, while the spatial environment is represented with a *location graph*. On this location graph, nodes are used to indicate conflict zones, camps, and other settlements of note. The nodes are interconnected using edges, which represent roads in most cases (and similarly-structured walking routes in a few exceptional cases). The location graph is represented as an adjacency list: nodes are resolved as Python objects, each holding a list of their neighbouring nodes, all arranged as a list. Suleimenova et al. [45] provide a detailed description of the algorithm, including the flowcharts used for agent-decision making as well as a detailed description of the "conflict" ruleset. Within this paper, we also introduce a second "tension" ruleset in some of our scalability benchmarks, which we describe in further detail in Section 5.4.

When conflicts erupt, the agents rapidly move away from the conflict zones in search of safe havens (typically camps). This means that macroscopically the agent movement patterns tend to have complex spatial characteristics and are of highly varying intensities over time, while the interaction between agents is only indirect. For instance, the arrival of an agent in a camp may increase its occupancy, which could then lead to a reduced likelihood of other agents choosing that camp as a destination.

Most of the existing parallel ABS codes use a spatial distribution of agents, with processes calculating sub-parts of the spatial domain and only the agents residing there. In particular, RepastHPC has been shown to run efficiently across up to 32K cores for certain basic use cases using this approach [10]. In our case, such an approach is not entirely practical because the spatial movement patterns of our agents are complex, fast, and often directed. This means that agents would very frequently traverse (spatial) process boundaries, which in turn leads to major load balancing issues. We have designed an approach that uses a location and agent parallelisation that is not based on spatial criteria, but maintains the same (equal) agent and location distribution throughout the simulation. Within Flee such an approach is possible, because agents only interact indirectly with each other (through updates of location states).

### 3.1 Outline of the PFlee Algorithm

Flee requires a range of input parameters, which specify the environment and population of agents along with their properties. The population is parameterized by the initial number of refugee agents $N$ and the number of new agents inserted during the current time step $N_{new}$. We note that a time step in Flee-based simulations corresponds to a day, we therefore use the terms *time step* and *day* interchangeably in the remaining of this document. The pace of agents' movement is governed by a maximum move speed per day $v_{\max}$ and an awareness level $A$, which is the distance (measured in number of link hops) an agent takes into account when choosing a destination. The environment is modelled by an attributed weighted graph of routes between locations $G_{\mathcal{L}} := (V_{\mathcal{L}}, E_{\mathcal{L}})$, called *location graph*. In this graph, each vertex $\ell \in V_{\mathcal{L}}$ represents location and has a tuple of attributes including location type and conflict date. Each edge $e := (u, v) \in E_{\mathcal{L}}$ corresponds to a direct route connecting two locations $u$ and $v$ from $V_{\mathcal{L}}$ and has a positive-valued weight $d_G(e) \in \mathbb{R}^+$ equal to the route distance between $u$ and $v$ measured in kilometers. Note that links of the location graph may be one or two-way and, hence, $G_{\mathcal{L}}$ may be directed or undirected depending on the route system between locations.

A sequential version of the Flee algorithm is described in Algorithm 1. In this Flee pseudocode, lines 1-12 correspond to the `evolve()` function, which is the core function of the Flee algorithm. The simulation starts at line 13, which corresponds to the extraction of the geospatial information, such as details about location – cities and camps – or border closures, required for the simulation. In line 14, refugees are assigned to the locations. Afterward, in lines 15-24, Flee starts to assign and relocate the refugee agents each day among all available camps on that day. In a nutshell, for each day, new agents are added to a random conflict location (lines 16-20). Then, agents are relocated to new locations (lines 22-23). The probability of relocating is determined by the move chance, which is location-dependent. In lines 2-7, during the execution of the `evolve()` function, an agent moves to a new location if a new location is selected by a path selection process. By moving to a new location, the refugee population for the old and new locations is updated. In lines 8-11, if the agent traveled with a lower move speed on a particular day, then a new move chance calculation (and possible move) is performed. The agent's move speed is no more than 200 km/day[2].

A core assumption in Flee (both sequential and parallel) currently is that agents do not communicate with each other directly. Instead, the behavior of agents is partially determined by the state of locations, and in turn the behavior of agents can lead to a change in location state as well. For example, if an agent arrives in a camp that is close to capacity, then the state of the camp location will change such that it is even closer to its capacity (or in some cases even reaching its capacity).

This simplifying assumption greatly reduces the computational burden of Flee and simplifies the parallelization scheme, which is described in detail in the following subsections. However, if one were to use this parallelisation scheme in a system where agents do directly interact with each other, additional collective MPI operations would be required during each time step. Essentially, our parallelisation scheme works very well for the Flee code, but for agent-based systems where agents heavily interact directly with each other on a local level, a more traditional parallelisation with a spatial agent decomposition (as done e.g. in RePast HPC) will result in higher performance.

In the next subsections, we show how to parallelize most of the computation using a parallel approach and present a parallel version of Flee algorithm, called PFlee. We do the parallelization at two levels, namely Agent parallelization (also called "classic mode") and Agent+space parallelization (also called "advanced mode" or ASP).

#### *3.1.1 Agent parallelization*

We start by explaining our basic approach to parallelize agent-decision making which is the most expensive calculation component in our code. This approach is the central component of PFlee and is applied both in this basic algorithm and the more sophisticated ASP approach. We parallelize the decision-making by distributing the agents evenly across the processes. This is done through a simple modification of the `addAgentToConflictLocation()` function, presented in Algorithm 2. In this parallelization approach, the location graph is replicated on each process.

In its simplest form, it is possible to only use agent-parallelization. In agent parallelization, agents are distributed evenly among processes. Therefore, each process holds the same number of Python objects of the `Person` class of Flee.. The parallel version of the `evolve()` function, which propagates the whole system by one timestep, works as follows:

1) Update location scores (which determine the attractiveness of locations to agents). This is equivalent to line 2 in Algorithm 1.
2) Evolve all agents on the local process, using the Flee ruleset. This is equivalent to lines 4-5 in Algorithm 1.
3) Aggregate the total number of agents per location across processes, using one or more `MPI_Allreduce` functions. This is a parallelized version of the operation in line 6 in Algorithm 1.
4) Complete the travel, for agents that have not done so already. This corresponds to lines 9-10 in Algorithm 1.
5) Aggregate the total number of agents per location across processes, using one or more `MPI_Allreduce` functions. This is necessary in this parallelized version of Flee, to have a global view of the agents in the simulation, before moving to the next time step.

The total number of agents is aggregated using the `MPI_Allreduce` collective operation. This can either be done on a location-by-location basis (low latency mode), resulting in one small `MPI_Allreduce` call for each location; or in bulk, which requires a packing operation on each process, followed by a single `MPI_Allreduce` to synchronize all locations and an unpacking task (high latency mode). In most but not all cases, the high latency mode is more efficient than the low latency mode.

We note that the aggregation of the number of agents for every location is a necessary step before an agent moves (i.e., in the case of moving an agent and in the case of examining whether an agent has finished their travel), as the rules that determine agent movements depend on the location scores (updated within the `updateLocationScores` operation), which in turn depend on the number of agents on each location (updated when the `updateLocationInfo` function is executed), along with other parameters. Although this approach works well with a large number of agents and a small location graph, the lack of location parallelization can become a bottleneck for larger and medium-sized location graphs.

#### *3.1.2 Agent+space parallelization (ASP) and other parallelization optimizations*

To add an extra layer of parallelization to the computations in Flee, we developed a more advanced algorithm that distributes the responsibility of location updates across the different processes. The parallel location update is depicted in Algorithm 3. In this version of the algorithm, we distribute the list of locations, i.e. the vertices of the location graph $G_{\mathcal{L}}$ evenly across all processes (lines 1-7). Therefore, each process holds the same number of Python objects of the `Location` class of Flee. Note that, the location graph in PFlee is an adjacency list, therefore each vertex also holds a list of neighbouring vertices. Location states, i.e. location scores, are replicated across processes, as they determine the movement of agents. Each process updates the locations assigned to it locally. Once all locations are updated, we synchronize all the location scores across all processes (line 12), which requires one `MPI_Allgatherv` operation. This operation is performed only once in the parallel version of the `evolve` function, in the `updateLocationScores` operation of Algorithm 1, and within step 1, as described in Section 3.1.1. This operation is necessary so that all locations have the updated location scores for their neighboring locations, which may belong to different processes.

## 3.2 Computational complexity

Besides Flee pseudocode, Algorithm 1 presents a computational complexity analysis for computational complexity

2. Authors have performed a range of sensitivity tests on agent's move speed and found that the simulation error increases when they choose lower move speed limits while higher move speeds have lower sensitivity on the simulation output [45].

**Algorithm 1** Pseudocode and complexity analysis for the FLEE simulation algorithm.

**Input:**
- $D$ : number of days for simulation
- $N := N^{(0)}$ : initial number of refugee agents (at day $d = 0$ of simulation)
- $N_{new} := \mathbb{E}[\Delta N^{(d)}]$ : average daily increase in the number of agents
- $v_{\max}$ : maximum move speed per day
- $A$ : awareness level
- $G_{\mathcal{L}} := (V_{\mathcal{L}}, E_{\mathcal{L}})$ : location graph representing environment

```
                                                                              cost                                                    times
 1: function evolve( $agents$, $G_{\mathcal{L}}$, $A$)
 2:     updateLocationScores($G_{\mathcal{L}}$)                               $\mathcal{O}\left(|V_{\mathcal{L}}|\Delta^-(G_{\mathcal{L}})\right)$
 3:     for each $agent$ in $agents$                                                                                                  $\sum_{d=1}^{D} N^{(d)}$
 4:         $new\_loc$ ← pathSelection($agent, G_{\mathcal{L}}, A$)           $\mathcal{O}\left(\delta(\ell|A, G_{\mathcal{L}})\right)$
 5:         moveAgent($agent, new\_loc$)                                      $\mathcal{O}(1)$
 6:         updateLocationInfo($new\_loc$)                                    $\mathcal{O}(1)$
 7:     end for
 8:     for each $agent$ in $agents$                                                                                                  $\sum_{d=1}^{D} N^{(d)}$
 9:         finishTravel($agent, G_{\mathcal{L}}, A$)                         $\mathcal{O}\left((S-1)\cdot\delta(\ell|A, G_{\mathcal{L}})\right)$
10:         updateLocationInfo($new\_loc$)                                    $\mathcal{O}(1)$
11:     end for
12: end function


13: Extract locations (conflict zones, camps, etc) and routes information from input files
14: $agents$ ← AddInitialRefugees($day = 0, N$)                               $\mathcal{O}(N)$                                        1
15: for all day $d \in [1 \dots D]$                                                                                                   $D$
16:     AddNewConflictZones($day = d, G_{\mathcal{L}}$)                       $\mathcal{O}(|V_{\mathcal{L}}|)$
17:     $new\_agents$ ← DailyNewRefugees($day = d, N_{new}$)                  $\mathcal{O}(\Delta N^{(d)})$
18:     for each $agent$ in $new\_agents$                                                                                             $\sum_{d=1}^{D} \Delta N^{(d)}$
19:         addAgentToConflictLocation($agent, \ell$)                         $\mathcal{O}(1)$
20:     end for
21:     $agents$ ← $agents$ + $new\_agents$                                   $\mathcal{O}(\Delta N^{(d)})$
22:     enactBorderClosures($day = d, G_{\mathcal{L}}$)                       $\mathcal{O}(|E_{\mathcal{L}}|)$
23:     evolve($agents$, $G_{\mathcal{L}}$, $A$)
24: end for
```

**Algorithm 2** Parallel agent distribution in Flee

```python
def addAgentToConflictLocation(self, location):
  self.total_agents += 1
  if self.total_agents % self.mpi.size == self.mpi.rank:
    self.agents.append(Person(location))
```

**Algorithm 3** Parallel location update in Flee

```python
locations_per_rank = int(len(locations) / mpi.size)
lpr_remainder = int(len(locations) % mpi.size)

offset = int(mpi.rank) * int(locations_per_rank) + int(min(
    ↪ mpi.rank, lpr_remainder))

if mpi.rank < lpr_remainder:
    locations_per_rank += 1

for i in range(offset, offset + locations_per_rank):
    locations[i].updateAllScores(self.time)

synchronize_locations(offset, offset + locations_per_rank)
```

of its building blocks. In this analysis, we denote $N^{(d)}$ a number of refugees and $\Delta N^{(d)} = N^{(d)} - N^{(d-1)}$ an increase in refugee amount at day $d$. We also assume that the number of refugees monotonically increases as the conflict develops, thus, $\Delta N^{(d)} \geq 0$.

The computation of the PFlee involves two major contributors to the overall complexity: (i) agents computing new directions to move and (ii) locations updating information on agents arrival. The former dominates in the computational costs, while the latter defines the communication overhead of the parallel implementations.

In order to decide on the new direction to move (line 11), each agent should evaluate a set of paths starting at the current location $\ell$. The number of paths to evaluate $\delta(\ell|A, G_{\mathcal{L}})$ generally depends on the awareness level of the agent and structure of the location graph $G_{\mathcal{L}}$. On the one hand, $\delta(\ell|A, G_{\mathcal{L}})$ is bounded by the number of possible destinations in the location graph $|V_{\mathcal{L}}| - 1$. On the other hand, its quantity cannot exceed the total number of paths from $\ell$ with a length less than equal to $A$. Since agent with no awareness selects future destination randomly in $\mathcal{O}(1)$ time, we presume $\delta(\ell|0, G_{\mathcal{L}}) = 1$. Taking into account that the total number of paths of length $a$ from the given location $\ell$ always remains below $\Delta^-(G)^a$ for any graph $G$, where $\Delta^-(G)$ is a maximum out-degree in $G$, we obtain for $A > 0$:

$$\delta(\ell|A, G_{\mathcal{L}}) \leq \sum_{a=1}^{A} \Delta^-(G_{\mathcal{L}})^a = \frac{\Delta^-(G_{\mathcal{L}})}{\Delta^-(G_{\mathcal{L}}) - 1}(\Delta^-(G_{\mathcal{L}})^A - 1)$$

and, hence, $\delta(\ell|A, G_{\mathcal{L}}) = \mathcal{O}\left(\min\left\{|V_{\mathcal{L}}|, \Delta^-(G_{\mathcal{L}})^A\right\}\right)$. Note that in certain situations, agent can target $S > 1$ hops in a single day (line 9 of Algorithm 1). In these cases, we must repeat the decision-making process for such agents $S - 1$ times to finish a day simulation. Even though the theoretical maximum of $S$ is upper bounded by the ratio of the maximum movement speed to the length of the shortest link between locations $v_{\max}/\min_{e\in E_{\mathcal{L}}} d_G(e)$, which can be high, in practice, there are seldom agents that will perform more than two hops in a single time step, due to the assumptions currently used in the code. Therefore, for the performance modelling purposes, one can usually assume that $S < 3$, depending on the chosen ABM rule set, and, thus, $S$ can be excluded from the complexity analysis as a small constant factor. Under this assumption, we estimate the contribution of an agent to the computational complexity per simulation day by

$$\mathcal{O}\left(S\cdot\delta(\ell|A,G_{\mathcal{L}})\right)=\mathcal{O}\left(\min\left\{|V_{\mathcal{L}}|,\Delta^{-}(G_{\mathcal{L}})^{A}\right\}\right) \quad (1)$$

By denoting $N = N^{(0)}$ as the initial number of agents and $N_{new} = \mathbb{E}[\Delta N^{(d)}]$ as the average daily increase in the number of agents, after summing up contributions (1) over all agents and $D$ simulation days, we obtain the following approximation for the time complexity of Flee simulation:

$$\mathcal{O}\left(\sum_{d=1}^{D} N^{(d)}\max_{\ell\in V_{\mathcal{L}}}\delta(\ell|A,G_{\mathcal{L}})+\sum_{d=1}^{D}\mathcal{O}\left(|V_{\mathcal{L}}|\Delta^{-}(G_{\mathcal{L}})\right)\right)=$$
$$\mathcal{O}\left(\max_{\ell\in V_{\mathcal{L}}}\delta(\ell|A,G_{\mathcal{L}})\left(DN+\sum_{d=1}^{D}(D-d+1)\Delta N^{(d)}\right)\right)$$
$$+\,\mathcal{O}\left(D|V_{\mathcal{L}}|\Delta^{-}(G_{\mathcal{L}})\right)=$$
$$\mathcal{O}\left(D\left(N+\frac{D+1}{2}N_{new}\right)\min\left\{|V_{\mathcal{L}}|,\Delta^{-}(G_{\mathcal{L}})^{A}\right\}\right)$$
$$+\,\mathcal{O}\left(D|V_{\mathcal{L}}|\Delta^{-}(G_{\mathcal{L}})\right)$$

Since in practice the overall number of new agents in the simulation $N^{(D)} - N$ rarely exceeds the initial number of agents $N$, or equivalently $N_{new} = \mathcal{O}(N/D)$, the total computational complexity can BE further simplified to

$$\mathcal{O}\left(D\left(N\times\min\left\{|V_{\mathcal{L}}|,\Delta^{-}(G_{\mathcal{L}})^{A}\right\}+|V_{\mathcal{L}}|\Delta^{-}(G_{\mathcal{L}}\right)\right)$$

This formula has a simple interpretation: computational costs of the Algorithm 1 are proportional to the number of simulation steps, the number of agents, and the complexity of the decision rules which are determined by the structure of the location graph.

For the agent-parallel version, the number of agents $N$ in the above formula is replaced by $\lceil N^{(d)}/p\rceil$, where $p$ is the number of cores/processes used. Additionally, for the agent-space parallel version, the number of nodes $|V_{\mathcal{L}}|$ is replaced by $\lceil |V|_{\mathcal{L}}/p\rceil$, where $p$ is the number of cores/processes used.

### 3.3 Communication costs

The main communication and synchronization logic is encapsulated in the method `evolve` of the object instances from class `Ecosystem`. This method is called at each iteration of the PFlee algorithm. It includes the following steps that require collective communication operations: synchronization of locations (method `synchronize_locations`), synchronization of spawn counts, and updating agent counts (method `updateNumAgents`). The `evolve` method calls the `synchronize_locations` method in the ASP mode (`loc-par` parallel mode). This operation uses `MPI_Allgatherv` to merge chunks of double-precision floating point arrays of size $k\cdot(|V_{\mathcal{L}}|/p)$ from each process into a single array, where $k$ denotes the number of scores per location. Synchronization of spawn counts requires a single call of `MPI_Allreduce` for integer arrays with $|V_{\mathcal{L}}|$ elements. `evolve` calls updated the number of agents twice: after evolving agents (select path and move) and after finishing agent traveling. In the `high_latency` mode, `updateNumAgents` requires a single call of `MPI_Allreduce` for integer arrays with the total number of elements equal to:

$$\sum_{\ell\in V_{\mathcal{L}}}(1+|\text{links}(\ell)|+|\text{closed\_links}(\ell)|)\le|V_{\mathcal{L}}|+|E_{\mathcal{L}}|.$$

The communication costs are summarized in Table 1.

| MPI collective | Data type | Chunk size | Times |
|---|---|---|---|
| `MPI_Allgatherv` | DOUBLE | $k\cdot\lvert V_{\mathcal{L}}\rvert/P$ | $D$ |
| `MPI_Allreduce` | INT | $\lvert V_{\mathcal{L}}\rvert$ | $D$ |
| `MPI_Allreduce` | INT | $\lvert V_{\mathcal{L}}\rvert+\lvert E_{\mathcal{L}}\rvert$ | $2D$ |

TABLE 1: Communication costs of the parallel algorithm

## 4 Numba-based performance optimization

To elevate the performance of PFlee beyond the limits of the standard Python interpreter, we employ the Numba [37] python library to optimize hotspots. A preliminary performance analysis of Flee allowed us to identify the critical parts of the code that jointly consume more than 35% of the total runtime. These parts refer to three functions of the Flee code and are mainly associated with generating random values with probability support.

More precisely, all selected functions are responsible for calculating the probability that a moving agent will select a given route, and for making the probabilistic route selection. In this case, the 1st function, called `CalculateLinkWeight` determines the weights of each adjacent link. The 2nd function (called `NormalizeWeights`) normalizes the weights. The last `RandChoice` function returns the eventual movement decision made. Table 2 indicates the percentage of total computation time that the Flee application spent executing a given Kernel for South Sudan case using a single node with two AMD EPYC 7742 processors.

| Function | 1 | 2 | 3 | In total |
|---|---|---|---|---|
| Percentage [%] | 16.80 | 8.41 | 10.68 | 35.89 |

TABLE 2: Percentage of the total execution time of hotspots measured for the case of South Sudan on a single node with two AMD EPYC 7742 CPUs

All functions are eligible for performance improvements by Numba, allowing their compilation processes to succeed. Every selected part of code is marked as Numba functions and translated into machine codes during application execution (just-in-time compilation). As a result, every function compiled once is used tens of millions of times in a typical simulation run.

To examine the proposed approach, we utilize a single node with two AMD EPYC 7742 processors and the case of South Sudan. Enabling Numba-based modification for PFlee reduces the total execution time and improves the overall performance achieving a speedup of about 1.44x. The applied analysis also indicates a significant performance improvement for the 2nd and 3rd functions. In this case, employing Numba leads to about 10.8x and up to 11.6x faster execution of the `NormalizeWeights` and `RandChoice` functions, respectively. In contrast, the 1st function with Numba enabled accelerates computations of about 1.1x.

## 5 Performance evaluation

We evaluate the PFlee agent-based modeling code under different synthetic and real-world execution scenarios, on two supercomputers, with different architectures:

- **Hawk** at *HLRS*[3] consists of 5632 compute nodes, two AMD EPYC Rome 77242 CPUs of 64 cores each, at 2.25GHz, and 256GB of RAM, interconnected with InfiniBand HDR.
- **Altair** at *PSNC*[4] consists of 1320 compute nodes, with two Intel Xeon Platinum 8268 CPUs of 24 cores each, at 2.9GHz, with 192 GB RAM each, interconnected with InfiniBand EDR.

On Hawk, PFlee runs using Python 3.8.3 and MPI MPT 2.23, used with mpi4py 3.0.3 for parallelization. On Altair, PFlee runs with Python 3.7.3 and OpenMPI 4.0.0, used with mpi4py 3.0.3. Additionally, the Flee code uses the Python libraries NumPy (1.19.0 on Hawk, 1.18.1 on Altair), SciPy (1.5.0 on Hawk, 1.4.1 on Altair), and Numba 0.55.1 for the optimized version of the code.

### 5.1 Datasets

#### *5.1.1 Synthetic datasets*

In our evaluation, we use synthetic inputs to exhaustively evaluate the scalability of PFlee. Real-world datasets refer to specific scenarios where the location graph and the number of agents are preset. Our synthetic inputs include synthetically generated location graphs, which are 2-dimensional grids of locations. The synthetic location graphs are regular (with a constant node degree) and the weights of the graph are randomly generated. The synthetic graphs are denoted by $V_1 - V_2 - v_d$, where $V_1 \times V_2$ are the nodes of the graph and $v_d$ is the constant node degree. In these synthetic datasets, we also set the initial number of agents $N$, which are initially randomly distributed to the nodes of the graph denoted as conflict locations. New conflict zones are added at various timesteps, at existing locations of the graph.

#### *5.1.2 Real datasets*

For the real datasets, we used two conflict scenarios: the cases of *South Sudan* and *Nigeria*. The South Sudan civil war started in December 2013 between forces of the government and opposition forces. About 400,000 people were estimated to have been killed in the war by April 2018 and more than 2.3 million people have been displaced forcibly since the start of the conflict. For this scenario, because the situation suddenly got worse during this period, we studied 426 days between 1st of July 2016 till 31st of June 2017, covering almost one million refugees. In the extracted location graph from the available data sources, such as UNHCR data portal, ACLED (Armed Conflict Location and Event Data Project), and OSM (Open Street Map), there are 76 conflict locations, 18 camps in neighboring countries, 39 towns, and 171 routes between them.

In 2014, violent attacks by the Islamist group Boko Haram started to spill over Nigeria's north-eastern frontier and the neighboring countries, Cameroon, Chad, and Niger, which have been drawn into what has become a devastating regional conflict. For this conflict scenario that forced over 3.2 million people to displace, the simulation period starts on 1st of March 2016 and ends on 20th of April 2021 (1887 days) because the number of refugees in neighboring countries surged in the first quarter of 2016. The extracted location graph of this scenario includes 47 conflict locations, 8 camps, 3 towns, and 64 routes between them. Furthermore, more than 300,000 agents' (refugees') movements were simulated in this scenario.

### 5.2 Impact of parallelization and optimization

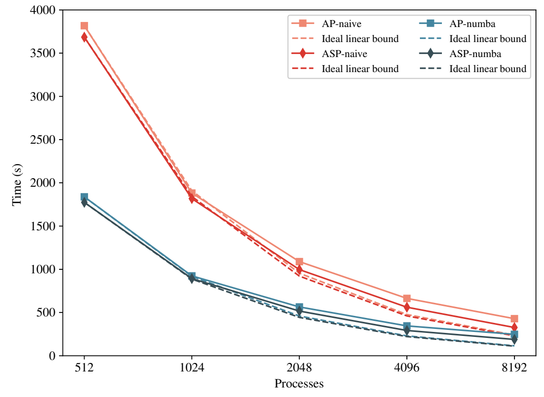
We first evaluate the agent-parallel (AP) and agent-space-parallel (ASP) parallelization modes of Flee, as well as the performance impact of using the Numba library in Python, on a synthetic, large-scale simulation scenario, with 100 million agents ($N = 10^8$) on the synthetic $50 - 50 - 4$ graph, for 100 epochs, on 4 up to 64 nodes on Hawk and 4 up to 32 nodes Altair. Note that the cores per node are different on the two systems, thus simulations on Hawk correspond to 512 up to 8192 cores, while simulations on Altair correspond to 192 up to 1536 cores. The results in Figure 2 (x-axis is logarithmic) first indicate that using Numba boosts the performance of PFlee on both systems. Numba manages to decrease execution time by about 50%. Additionally, the ASP mode significantly outperforms the AP mode, due to location parallelization and the more efficient bundling of messages. The performance gains become more evident as the number of cores increases; ASP improves PFlee scalability. For the particular execution scenario, on 32 nodes, the performance improvement from location parallelization is 20% on both systems.

### 5.3 Scalability analysis using synthetic datasets

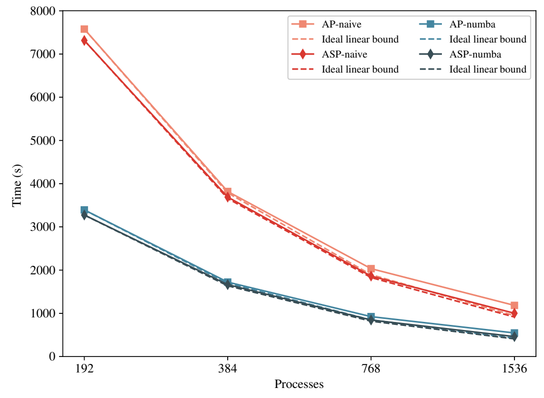
We subsequently evaluate the scalability of PFlee using synthetic datasets, on 4 up to 64 nodes on Hawk and Altair. We focus on the ASP parallelization mode of PFlee and use the Numba library. We examine two different cases. In the first case, we use a synthetic $50 - 50 - 4$ graph as the input and vary the initial number of agents in the simulation. In the second case, we fix the initial number of agents in the simulation to 100 million ($N = 10^8$) and perform simulations with different synthetic graphs as input. In both cases, no new agents are added in the simulation at any subsequent epoch, and simulations run for 100 epochs each.

Figure 3 (both axes are logarithmic) demonstrates the results for the two cases. In Figure 3a, we vary the number of

3. HLRS is the High-Performance Computing Center Stuttgart, in Germany.
4. PSNC is the Poznan Supercomputing and Networking Center, in Poland.

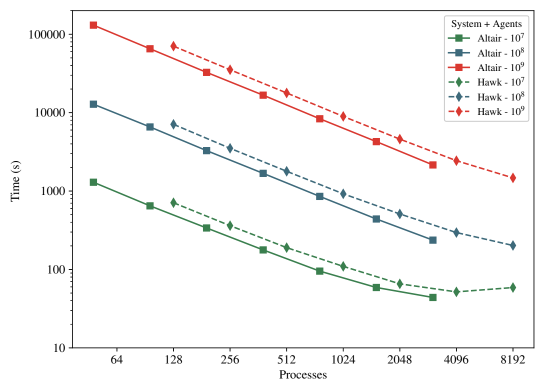


(a) Hawk (128 processes per node)

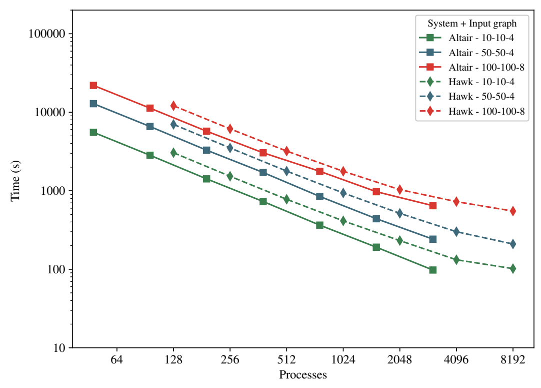


(b) Altair (48 processes per node)

Fig. 2: Comparison between the agent-parallel (AP) and agent-space parallel (ASP) modes of PFlee, with and without Numba ('numba' / 'naive'), on a synthetic 50-50-4 graph with 100 million agents, for 100 epochs, on *Hawk* and *Altair*.



(a) Varying the number of agents on a synthetic $50-50-4$ input graph



(b) Varying the input graph on a fixed number of $N = 10^8$ agents

Fig. 3: Evaluating the effect of varying numbers of agents and the effect of varying the input graph on the execution time of PFlee, for 100 epochs, on *Hawk* and *Altair*.

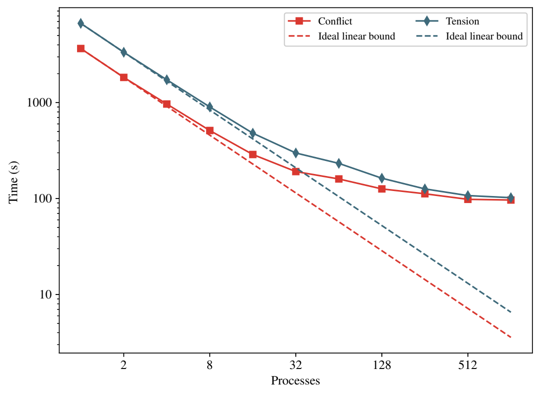
initial agents from 10 million to 1 billion, with the $50-50-4$ graph as input. First, we observe that execution time is proportional to the number of agents in the simulation. Second, we observe that, in both systems, scalability is better with higher numbers of agents, since the higher number of agents impacts the computation-to-communication ratio of the simulation, in favor of computation. This is also evident in the case of 10 million agents, where scalability breaks on both Hawk and Altair when the number of nodes increases to more than 16. Additionally, while Altair offers better execution times on lower numbers of processes than Hawk, in the case of 10 million agents, its performance deteriorates faster because of more inter-node communication. This effect occurs due to the lower number of cores per node on Altair, as well as due to the reduced computational work per process, as Altair cores are faster.

In Figure 3b, we plot the performance of PFlee for 100 million agents, with three different input graphs, $10-10-4$, $50-50-4$, and $100-100-8$, with 100, 2500, and 10000 nodes respectively. First, we observe that, as in the case of scaling the number of agents, the size of the graph impacts the total execution time, however, the increase is not proportional to the size of the graph (in number of graph nodes). For the smaller graph, we observe excellent scalability for PFlee on Altair, while on Hawk, we observe that the scalability reduces when we move from 32 to 64 nodes, for the two smaller graphs, $10-10-4$ and $50-50-4$. For the larger $100-100-8$ graph, we observe less scalable execution of PFlee on both systems, however, we also note that execution time on Altair is increased for more than 16 nodes, where inter-node communication becomes more impactful in the total execution time. In practice, larger graphs favor the selection of Hawk, where the high number of cores per node helps in containing the more costly inter-node communication. Finally, we do not observe any breaks in the scalability of PFlee.

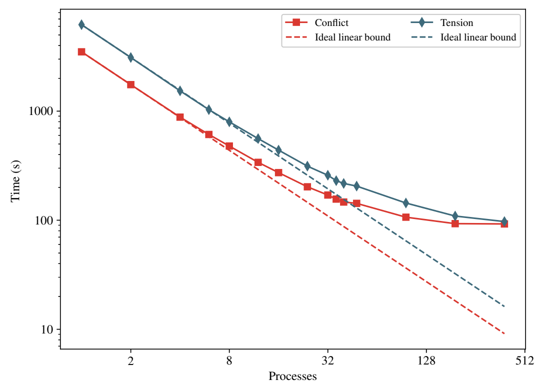
## 5.4 Scalability analysis using real datasets

We finally evaluate the performance of PFlee on Hawk and Altair for the two real datasets, namely the cases of South Sudan and Nigeria, to assess the execution time and scalability of Flee on these and similar realistic scenarios. We also leverage the real datasets to showcase the flexibility of the ruleset implementation in Flee. Figures 4 and 5 demonstrate the execution time of PFlee on up to 8 nodes on Hawk (1-1024 cores) and Altair (1-384 cores), using the following two rulesets:

- **Conflict**: The base ruleset, as described in [45] and in Section 3.
- **Tension**: A "tension-migration" ruleset, where agents have a much wider awareness of their surroundings, and where conflict zones are replaced by tension zones, which have a lower move chance, i.e. lower probability for an agent traversing a link. Normal towns also have a reduced ruleset. In this case, the awareness level $A$ is

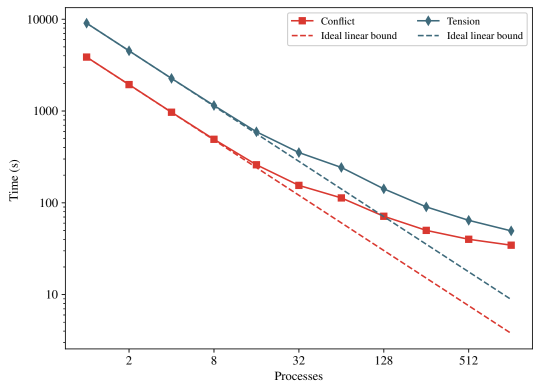


(a) Hawk (128 processes per node)

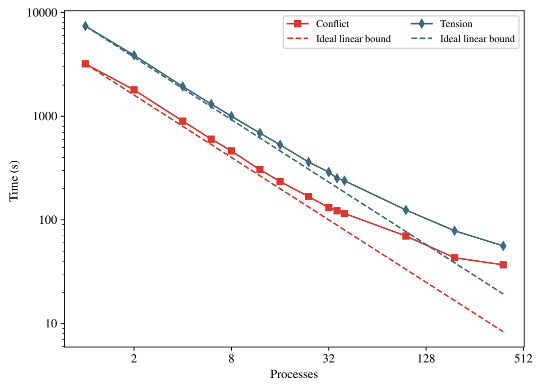


(b) Altair (48 processes per node)

Fig. 4: Evaluating the execution time of PFlee for two different simulation scenarios for South Sudan; a full conflict migration scenario and a less immediate movement scenario, based on areas of tension rather than violent conflict, for the same duration of 426 days, on *Hawk* and *Altair*.



(a) Hawk (128 processes per node)



(b) Altair (48 processes per node)

Fig. 5: Evaluating the execution time of PFlee for two different simulation scenarios for Nigeria; a full conflict migration scenario and a less immediate movement scenario, based on areas of tension rather than violent conflict, for the same duration of 1887 days, on *Hawk* and *Altair*.

set to 5 instead of 1, leading to a larger computational complexity.

For both rulesets, we observe a similar trend in the scalability of PFlee, with a minor overhead on the execution time of the **Conflict** ruleset, attributed to the difference in the move chance between the two scenarios. We can thus safely assume that the parallel performance of PFlee is robust and independent of the ruleset. It is worth annotating that the South Sudan case involves a small number of agents (fewer than 1 million), therefore there is limited granularity per process and limited potential performance gains from scaling this particular scenario on more cores. Additionally, we observe similar performance trends in both systems.

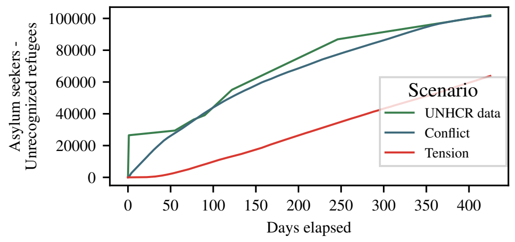
## 6 Scientific results (use case(s))

To investigate the behavior of PFlee from a migration research perspective, we present the predicted arrival rates for the two largest camps in each conflict, for South Sudan and Nigeria, in Figure 6. Both these scenarios are historical conflicts, where people were forced to leave their home locations due to violence. We provide the number of arrivals for each camp (according to UNHCR data) and compare this with the PFlee simulation results using the two rulesets. As we can see, the results from the conflict ruleset correspond well with the UNHCR data in the case of the largest camp in South Sudan (Rhino) and Nigeria (Diffa), and the conflict ruleset provides more accurate results when we look at the average differences across all four camps. However, because the code overpredicts arrivals for the two second-largest camps (Adjumani and Minawao), we observe that the tension ruleset leads to slightly more accurate results for those camps.

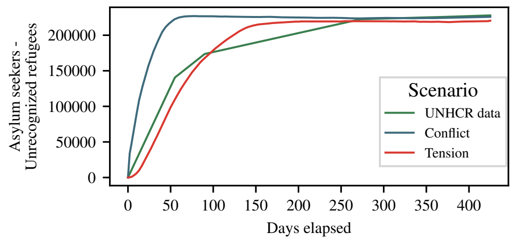
The quicker turnaround time offered by parallelization and HPC execution provides us with several benefits: (i) it provides us with the memory required to simulate with much higher numbers of agents (and more complex ones), (ii) it reduces the time to completion, allowing us to more quickly make forecasts, and (iii) it enables us to experiment with more advanced decision-making algorithms with minor impact on the time completion. In addition to the benefits of parallel execution, HPC environments are also ideal to facilitate ensembles of many simulations, for instance, to forecast across a wide range of conflict scenarios or to assess the sensitivity of the main results to specific assumptions in the code.

### 6.1 Reuse potential

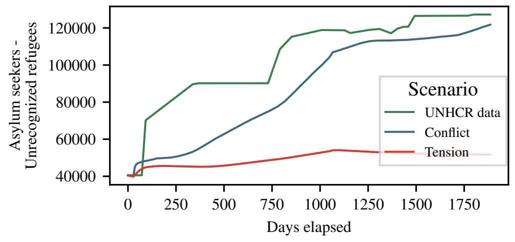
The main purpose of the Flee model is to provide forecasts of the arrivals of forcibly displaced human populations. To do this, the code relies on an iterative agent-based decision-making algorithm and a location graph which consists of locations (vertices) and routes (edges). The amount of direct interactions between agents needs to be relatively infrequent



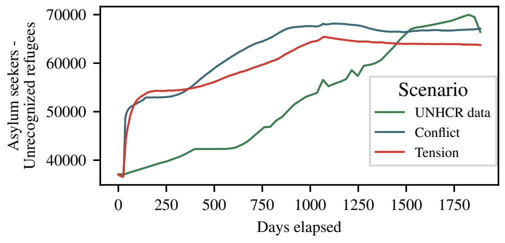
(a) Rhino camp in South Sudan



(b) Adjumani camp in South Sudan



(c) Diffa camp in Nigeria



(d) Minawao camp in Nigeria

Fig. 6: Comparison of arrivals between simulated results with PFlee with two scenarios, and UNHCR data, in four camps, Rhino and Adjumani, for the case of South Sudan, and Diffa and Minawao, for the case of Nigeria.

(e.g., once per time step) for the code to retain scalability. Indeed, currently for migration modeling, most effects are resolved indirectly through agents modifying the properties of specific locations and routes.

In terms of re-use, PFlee naturally lends itself well to related migration challenges, such as forecasting longer-term human migration or even the migration of other species. In these cases, the decision-making ruleset will need to be revised, as well as the properties of the location and path objects, but the parallelization approach can be retained. There are three examples where we are making such adaptations. First, we have made a modified version of Flee that incorporates food insecurity [7]; extending such an implementation to one that models migration primarily driven by starvation rather than conflict is relatively straightforward. Second, we are currently adapting the Flee algorithm to model the movements of goods between locations, in collaboration with the STAMINA consortium [5]. Third, in collaboration with ECMWF[6] we are adapting Flee to model migration driven by weather and climate effects.

PFlee may also be re-engineered for some other scientific purposes, although we have not undertaken research in these directions as of this time. Feasible areas could include for instance the modeling of trade, pandemics (in a large scale, approximate, manner), and the effects of education. PFlee is generally not an appropriate solution for modeling processes that are strongly communication-driven. Examples of these include the propagation of knowledge across communities and the spread of infectious diseases on a local, individualized level. To specifically address the latter challenge, we instead developed a different simulation kernel (the Flu And Coronavirus Simulator [33], [35]) which is better suited to efficiently calculate the outcomes of large numbers of interpersonal interactions.

## 7 Conclusions and Future Work

In this paper, we presented a parallelized human migration simulation tool, PFlee, that facilitates the understanding of mechanisms governing the movement of people. It also enables us to explore the simulation of societies as complex adaptive non-linear systems, which are difficult to study with classical mathematical equation-based models. We described two levels of parallelization in PFlee: agent and agent+space parallelization. We investigated the computational complexity of these parallelization schemes, and have shown that the complexity of the algorithm is largely dependent on the number of agents (current and new), the number of locations and simulated days, as well as the maximum movement speed. We presented a scalability study using two supercomputers and a large synthetic graph for varying numbers of simulation agents. We observed near-linear scalability in classic and advanced modes, sometimes up to 1,000s of cores. Moreover, we evaluated the execution time for two different South Sudan simulation scenarios: a full conflict migration scenario and a less immediate danger movement, observing significant efficiency up to 32 cores.

Going forward, we seek to focus on several areas: first, we aim to improve the output infrastructure to enable more sophisticated debugging and validation investigations, second we want to extend the ruleset so that we can more realistically model mixtures of refugees with IDPs in our simulations, third we want to incorporate the movements of other relevant objects in Flee where relevant, e.g. the movement of goods in situations where supply chains or trade movements may become relevant for the overall population dynamics. Lastly, we aim to apply Flee to directly contribute to the humanitarian support in ongoing conflicts and are currently in the process of doing so in the context of Ethiopia.

## References

[1] Abar, S., Theodoropoulos, G.K., Lemarinier, P., O'Hare, G.M.: Agent based modelling and simulation tools: a review of the state-of-art software. Computer Science Review **24**(nil), 13–33 (2017). https://doi.org/10.1016/j.cosrev.2017.03.001, https://doi.org/10.1016/j.cosrev.2017.03.001

5. https://twitter.com/stamina_project
6. https://www.ecmwf.int

[2] Barrett, C.L., Bisset, K.R., Eubank, S.G., Feng, X., Marathe, M.V.: Episimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In: 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (11 2008). https://doi.org/10.1109/sc.2008.5214892, https://doi.org/10.1109/sc.2008.5214892

[3] Barthelemy, J., Toint, P.L.: A stochastic and flexible activity based model for large population application to belgium (2015)

[4] Bhatele, A., Yeom, J.S., Jain, N., Kuhlman, C.J., Livnat, Y., Bisset, K.R., Kale, L.V., Marathe, M.V.: Massively parallel simulations of spread of infectious diseases over realistic social networks. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 689–694 (5 2017). https://doi.org/10.1109/ccgrid.2017.141, https://doi.org/10.1109/ccgrid.2017.141

[5] Bisset, K.R., Chen, J., Feng, X., Kumar, V.A., Marathe, M.V.: EpiFast. In: Proceedings of the 23rd International Conference on Supercomputing - ICS '09. p. nil (- 2009). https://doi.org/10.1145/1542275.1542336, https://doi.org/10.1145/1542275.1542336

[6] Blandin, N., Colglazier, C., O'Hare, J., Brenner, P.: Parallel python for agent-based modeling at a global scale. In: Proceedings of the 2017 International Conference of The Computational Social Science Society of the Americas. pp. 1–7 (2017)

[7] Campos, C.V., Suleimenova, D., Groen, D.: A coupled food security and refugee movement model for the south sudan conflict. In: International Conference on Computational Science. pp. 725–732. Springer (2019)

[8] Chao, D.L., Halloran, M.E., Obenchain, V.J., Longini, I.M.: Flute, a publicly available stochastic influenza epidemic simulation model. PLoS Computational Biology **6**(1), e1000656 (2010). https://doi.org/10.1371/journal.pcbi.1000656, https://doi.org/10.1371/journal.pcbi.1000656

[9] Collier, N., North, M.: Repast hpc: A platform for large-scale agent-based modeling. Large-Scale Computing pp. 81–109 (2012)

[10] Collier, N., North, M.: Repast hpc: A platform for large-scale agent-based modeling. Large-Scale Computing pp. 81–109 (2012)

[11] Collier, N., North, M.: Parallel agent-based simulation with repast for high performance computing. SIMULATION **89**(10), 1215–1235 (2013). https://doi.org/10.1177/0037549712462620, https://doi.org/10.1177/0037549712462620

[12] Collins, A.J., Frydenlund, E.: Agent-based modeling and strategic group formation: a refugee case study. In: 2016 Winter Simulation Conference (WSC). pp. 1289–1300. IEEE (2016)

[13] Cordasco, G., Scarano, V., Spagnuolo, C.: Distributed mason: A scalable distributed multi-agent simulation environment. Simulation Modelling Practice and Theory **89**, 15 – 34 (2018). https://doi.org/https://doi.org/10.1016/j.simpat.2018.09.002, http://www.sciencedirect.com/science/article/pii/S1569190X18301230

[14] Davidsson, P.: Agent based social simulation: A computer science view. Journal of artificial societies and social simulation **5**(1) (2002)

[15] Edwards, S.: Computational tools in predicting and assessing forced migration. Journal of Refugee Studies **21**(3), 347–359 (2008)

[16] Eubank, S.: Scalable, efficient epidemiological simulation. In: Proceedings of the 2002 ACM Symposium on Applied Computing. p. 139–145. SAC '02, Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/508791.508819, https://doi.org/10.1145/508791.508819

[17] Eubank, S., Guclu, H., Anil Kumar, V.S., Marathe, M.V., Srinivasan, A., Toroczkai, Z., Wang, N.: Modelling disease outbreaks in realistic urban social networks. Nature **429**(6988), 180–184 (May 2004). https://doi.org/10.1038/nature02541, https://doi.org/10.1038/nature02541

[18] Fiddian-Qasmiyeh, E., Loescher, G., Long, K., Sigona, N.: The Oxford handbook of refugee and forced migration studies. OUP Oxford (2014)

[19] Geiger, B.C., Jahani, A., Hussain, H., Groen, D.: Markov aggregation for speeding up agent-based movement simulations. In: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems. pp. 1877–1885 (2023)

[20] Germann, T.C., Kadau, K., Longini, I.M., Macken, C.A.: Mitigation strategies for pandemic influenza in the United States. Proceedings of the National Academy of Sciences **103**(15), 5935–5940 (2006). https://doi.org/10.1073/pnas.0601266103, https://www.pnas.org/content/103/15/5935

[21] Gilbert, N., Troitzsch, K.: Simulation for the social scientist. McGraw-Hill Education (UK) (2005)

[22] Gogolenko, S.: Large scale agent-based social simulations with high resolution raster inputs in distributed HPC environments. In: Resch, M.M., Kovalenko, Y., Bez, W., Focht, E., Kobayashi, H. (eds.) Sustained Simulation Performance 2018 and 2019. pp. 205–214. Springer (2020)

[23] Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: PowerGraph: Distributed graph-parallel computation on natural graphs. In: Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). pp. 17–30. USENIX, Hollywood, CA (2012), https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez

[24] Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: GraphX: Graph processing in a distributed dataflow framework. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). pp. 599–613. USENIX Association, Broomfield, CO (Oct 2014), https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez

[25] Groen, D., Arabnejad, H., Suleimenova, D., Edeling, W., Raffin, E., Xue, Y., Bronik, K., Monnier, N., Coveney, P.V.: Fabsim3: An automation toolkit for verified simulations using high performance computing. Computer Physics Communications **283**, 108596 (2023)

[26] Hattle, A., Yang, K.S., Zeng, S.: Modeling the syrian refugee crisis with agents and systems. UMAP Journal **37**(2) (2016)

[27] Hébert, G.A., Perez, L., Harati, S.: An agent-based model to identify migration pathways of refugees: the case of syria. In: Agent-Based Models and Complexity Science in the Age of Geospatial Big Data, pp. 45–58. Springer (2018)

[28] Hristova, D., Williams, M.J., Musolesi, M., Panzarasa, P., Mascolo, C.: Measuring urban social diversity using interconnected geo-social networks. In: Proceedings of the 25th International Conference on World Wide Web - WWW '16. p. nil (- 2016). https://doi.org/10.1145/2872427.2883065, https://doi.org/10.1145/2872427.2883065

[29] Jahani, A., Arabnejad, H., Suleimanova, D., Vuckovic, M., Mahmood, I., Groen, D.: Towards a coupled migration and weather simulation: South sudan conflict. In: Computational Science–ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part V. pp. 502–515. Springer (2021)

[30] Kravari, K., Bassiliades, N.: A survey of agent platforms. Journal of Artificial Societies and Social Simulation **18**(1) (2015), https://EconPapers.repec.org/RePEc:jas:jasssj:2014-71-2

[31] Kyrola, A., Blelloch, G., Guestrin, C.: GraphChi: Large-scale graph computation on just a PC. In: 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). pp. 31–46. USENIX Association, Hollywood, CA (Oct 2012), https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kyrola

[32] Liu, X.: Simulation of refugee flow in europe. In: 2017 5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT 2017). Atlantis Press (2017)

[33] Mahmood, I., Arabnejad, H., Suleimenova, D., Sassoon, I., Marshan, A., Serrano-Rico, A., Louvieris, P., Anagnostou, A., Taylor, S.J.E., Bell, D., Groen, D.: Facs: a geospatial agent-based simulator for analysing covid-19 spread and public health measures on local regions. Journal of Simulation **0**(0), 1–19 (2020). https://doi.org/10.1080/17477778.2020.1800422, https://doi.org/10.1080/17477778.2020.1800422

[34] Miller, J.H., Page, S.E.: Complex adaptive systems: An introduction to computational models of social life. Princeton university press (2009)

[35] Mintram, K., Anagnostou, A., Anokye, N., Okine, E., Groen, D., Saha, A., Abubakar, N., Islam, T., Daroge, H., Ghorbani, M., et al.: Calms: Modelling the long-term health and economic impact of covid-19 using agent-based simulation. Plos one **17**(8), e0272664 (2022)

[36] Nagel, K., Rickert, M.: Parallel implementation of the TRANSIMS micro-simulation. Parallel Computing **27**(12), 1611 – 1639 (2001). https://doi.org/https://doi.org/10.1016/S0167-8191(01)00106-5, http://www.sciencedirect.com/science/article/pii/S0167819101001065, applications of parallel computing in transportation

[37] Numba documentation (Version 0.55). https://numba.readthedocs.io/

[38] Perumalla, K.S., Seal, S.K.: Discrete event modeling and massively parallel execution of epidemic out-

break phenomena. SIMULATION **88**(7), 768–783 (2012). https://doi.org/10.1177/0037549711413001, https://doi.org/10.1177/0037549711413001

[39] Picascia, S., Yorke-Smith, N.: Towards an agent-based simulation of housing in urban beirut. In: International Workshop on Agent Based Modelling of Urban Systems. pp. 3–20. Springer (2016)

[40] Rousset, A., Herrmann, B., Lang, C., Philippe, L.: A survey on parallel and distributed multi-agent systems for high performance computing simulations. Computer Science Review **22**(nil), 27–46 (2016). https://doi.org/10.1016/j.cosrev.2016.08.001, https://doi.org/10.1016/j.cosrev.2016.08.001

[41] Schmitt, T.G., Kumar, S., Stecke, K.E., Glover, F.W., Ehlen, M.A.: Mitigating disruptions in a multi-echelon supply chain using adaptive ordering. Omega **68**, 185 – 198 (2017). https://doi.org/https://doi.org/10.1016/j.omega.2016.07.004, http://www.sciencedirect.com/science/article/pii/S0305048316304431

[42] Shih, C., Yang, C., Fukuda, M.: Benchmarking the agent descriptivity of parallel multi-agent simulators. In: International Conference on Practical Applications of Agents and Multi-Agent Systems. pp. 480–492. Springer (2018)

[43] Shun, J., Blelloch, G.E.: Ligra: A lightweight graph processing framework for shared memory. SIGPLAN Not. **48**(8), 135–146 (Feb 2013). https://doi.org/10.1145/2517327.2442530, https://doi.org/10.1145/2517327.2442530

[44] Sokolowski, J.A., Banks, C.M., Hayes, R.L.: Modeling population displacement in the syrian city of aleppo. In: Proceedings of the Winter Simulation Conference 2014. pp. 252–263. IEEE (2014)

[45] Suleimenova, D., Bell, D., Groen, D.: A generalized simulation development approach for predicting refugee destinations. Scientific reports **7**(1), 1–13 (2017)

[46] Suleimenova, D., Low, W., Groen, D.: An agent-based forced displacement simulation: A case study of the tigray crisis. In: Computational Science–ICCS 2022: 22nd International Conference, London, UK, June 21–23, 2022, Proceedings, Part IV. pp. 83–89. Springer (2022)

[47] Tatara, E., Collier, N., Ozik, J., Macal, C.: Endogenous social networks from large-scale agent-based models. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). p. nil (5 2017). https://doi.org/10.1109/ipdpsw.2017.83, https://doi.org/10.1109/ipdpsw.2017.83

[48] UNHCR: Figures at a Glance. Available at: https://www.unhcr.org/figures-at-a-glance.html (2020)

[49] Wang, H., Wei, E., Simon, R., Luke, S., Crooks, A., Freelan, D., Spagnuolo, C.: Scalability in the mason multi-agent simulation system. In: 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT). pp. 1–10 (2018)

[50] Wu, Y., Cai, W., Li, Z., Tan, W.J., Hou, X.: Efficient parallel simulation over large-scale social contact networks. ACM Trans. Model. Comput. Simul. **29**(2) (Apr 2019). https://doi.org/10.1145/3265749, https://doi.org/10.1145/3265749

[51] Xiao, J., Andelfinger, P., Eckhoff, D., Cai, W., Knoll, A.: A survey on agent-based simulation using hardware accelerators. ACM Comput. Surv. **51**(6) (Jan 2019). https://doi.org/10.1145/3291048, https://doi.org/10.1145/3291048

[52] Xue, Y., Li, M., Arabnejad, H., Suleimenova, D., Jahani, A., Geiger, B.C., Wang, Z., Liu, X., Groen, D.: Camp location selection in humanitarian logistics: A multiobjective simulation optimization approach. In: Computational Science–ICCS 2022: 22nd International Conference, London, UK, June 21–23, 2022, Proceedings, Part III. pp. 497–504. Springer (2022)

[53] Ye, P.j., Wang, X., Chen, C., Lin, Y.t., Wang, F.y.: Hybrid agent modeling in population simulation: Current approaches and future directions. Journal of Artificial Societies and Social Simulation **19**(1), 12 (2016)

[54] Ye, P., Wang, X., Xiong, G., Chen, S., Wang, F.Y.: Tidec: A two-layered integrated decision cycle for population evolution. IEEE Transactions on Cybernetics (2020)

[55] Yeom, J., Bhatele, A., Bisset, K., Bohm, E., Gupta, A., Kale, L., Marathe, M., Nikolopoulos, D., Schulz, M., Wesolowski, L.: Overcoming the scalability challenges of epidemic simulations on blue waters. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium. pp. 755–764 (2014)