# P-FLEE: An Efficient Parallel Algorithm for Simulating Human Migration

Petros Anastasiadis*, Sergiy Gogolenko†, Nikela Papadopoulou*, Marcin Lawenda‡
Hamid Arabnejad§, Alireza Jahani§, Imran Mahmood§ and Derek Groen§
*Computing Systems Laboratory, National Technical University of Athens, Greece, Email: panastas@cslab.ece.ntua.gr
†High Performance Computing Center Stuttgart, Stuttgart, Germany, Email: gogolenko@hlrs.de
‡Poznan Supercomputing and Networking Center, Poznan, Poland
§Department of Computer Science, Brunel University London, Uxbridge, UK

***Abstract*—With over 79 million people forcibly displaced, forced human migration becomes a common issue in the modern world and a serious challenge for the global community. The Flee is a validated agent-based social simulation framework for forecasting the population displacements in the armed conflict settings. In this paper, we present two schemes to parallelize Flee, analyze computational complexity of those schemes, and outline results for benchmarks of our parallel codes with the real-world and synthetic scenarios on four state-of-the-art systems including a new European pre-exascale system, Hawk. On all testbeds, we evidenced high scalability of our codes. It exceeds more than 16,384 cores in our largest benchmark with 100 million agents on Hawk. Parallelization schemes discussed in this work, can be extrapolated to a wide range of ABSS applications with frequent agent movement and lesser impact of direct communications between agents.**

***Keywords*-migration, forcibly displaced people, agent-based social simulation (ABSS), parallel and distributed agent-based systems (PDABS), high-performance computing (HPC)**

## I. INTRODUCTION

In today's world, the issue of human migration is of huge importance to the global community, with over 79.5 million people forcible displaced [1] and immigration policies being one of the major topics in the media. Accurate forecasts for the movements of forcibly displaced people (FDP) can assist governments and non-governmental organizations to evaluate effect of the humanitarian policies such as relocation of humanitarian resources between the refugee camps or border closing. One way to provide these forecasts is through agent-based social simulations (ABSS), which enables exploration of the societies as complex adaptive non-linear systems, difficult to study with classical equation-based models.

Flee is an ABSS framework for forecasting FDP movements, validated successfully for a range of geographically different conflicts [2]. In this paper, we discuss an extension of the Flee framework to its parallel version, called P-Flee. P-Flee is a scalable code for simulating complex scenarios of forced migration with high level of details, which enables modelers to focus on the complexity of the conceptual model without worrying about the execution time. We present algorithmic solutions behind P-Flee implementation, as well as compare performance and scalability of our codes on several cutting-edge HPC platforms in order to establish guidelines for choosing the most advantageous architectures for migration simulations among accessible. P-Flee is available from GitHub[1] under a BSD 3-clause license.

## II. RELATED WORK

Most of the existing literature about agent-based human migration simulations focuses on the social aspects and neglects discussing implementation details related to the computational performance and scaling of the applications [2], [3], [4], [5], [6]. As a rare exception in [7], Blandin et al. propose a parallel Python implementation for human migration which scales up to 7 billion agents at a global scale, however the micro-scale agent behaviors have been largely ignored. In contrast, P-Flee implements agent-based human migration simulation with sufficient scalability while keeping an account for the micro-level agent behavior.

In P-Flee, the spatial environment is represented by a network of interconnected locations. HPC-compliant agent-based codes, in which environment is modeled by networks, are broadly discussed in literature. RepastHPC [8], [9] and D-MASON [10] constitute two most popular general-purpose parallel and distributed agent-based systems (PDABS) suitable for such ABSS. Besides general-purpose PDABS, HPC community developed a number of domain-specific HPC-compliant ABSS codes for applications in epidemiology [11], [12], [13], [14], social networks modelling [15], economics and logistics [16], urban planning [17], [9], [18]. Finally, in many occasions, such ABSS can be efficiently implemented on top of highly optimized multi-core and distributed graph-parallel frameworks like PowerGraph, GraphX, GraphChi, and Ligra or parallel graph processing libraries like PBGL and SNAP [19].

Nevertheless, majority of these codes uses a spatial decomposition In case of human migration, such an approach is less practical because the spatial movement patterns of agents are complex, fast and often directed, giving rise to challenges in terms of load balancing, which are hardly solvable with the above-mentioned codes due to their design centered around network partitioning. Instead we developed an approach that is both spatially parallel and agent parallel.

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

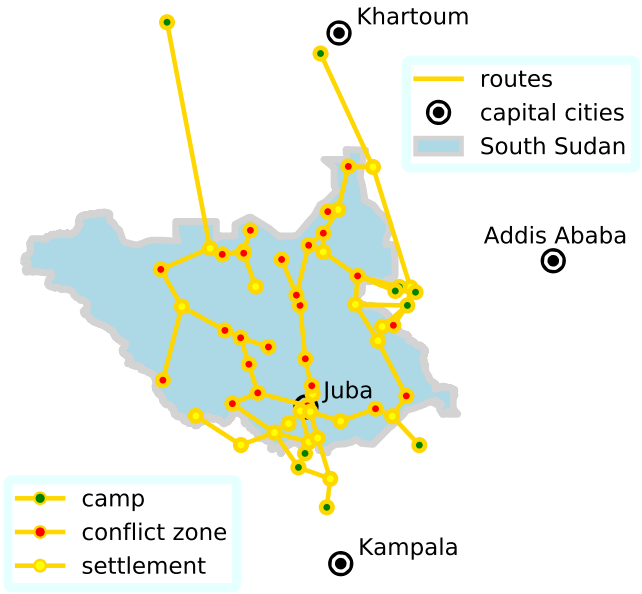



Figure 1. South Sudan conflict location graph (58 locations and 69 routes)

## III. The Flee Algorithm

### A. Outline of the Flee Algorithm

Flee requires a range of input parameters, which specify environment and population of agents along with their properties. The Flee code calculates the movement of displaced agents on a daily basis with a total number of simulation days $D$. Population is parametrized by the initial number of refugee agents $N := N^{(0)}$ and the average number of new agents inserted during the time step $N_{new} := \mathbb{E}[\Delta N^{(d)}]$, where $N^{(d)}$ denotes a total number of agents at day $d$. 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. Environment is modelled by an attributed weighted graph of routes between locations $G_{\mathcal{L}} := (V_{\mathcal{L}}, E_{\mathcal{L}})$, called *location graph* (cf. Fig. 1). In this graph, each vertex $\ell \in V_{\mathcal{L}}$ represents location and has a tuple of attributes including location type (conflict zones, camps, forwarding hubs and other major settlements) 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.

Algorithm 1 illustrates a sequential version of Flee. Suleimenova et al. [2] provide a detailed description of this algorithm including the flowcharts used for agent decision-making. Lines 11 and 16 constitute the most expensive component of Algorithm 1 and the main target for parallelization. They correspond to the decision-making where each agent selects further destination and path to move.

**Algorithm 1** Pseudocode for the Flee simulation algorithm

**Require:** $D$, $N$, $N_{new}$, $v_{\max}$, $A$, $G_{\mathcal{L}} := (V_{\mathcal{L}}, E_{\mathcal{L}})$
1: Extract locations and routes information from input files
2: AddInitialRefugees($day = 0, N$)
3: **for each** day $d \in [1 \dots D]$ **do**
4: AddNewConflictZones($day = d, G_{\mathcal{L}}$)
5: $new\_agents \leftarrow$ DailyNewRefugees($day = d, N_{new}$)
6: **for each** $agent \in new\_agents$ **do**
7: addAgentToConflictLocation($agent, \ell$)
8: **end for**
9: enactBorderClosures($day = d, G_{\mathcal{L}}$)
10: **for each** $agent \in$ system **do**
11: $\ell_{new} \leftarrow$ pathSelection($agent, G_{\mathcal{L}}, A$)
12: moveAgent($agent, \ell_{new}$)
13: updateLocationInfo($\ell_{new}$)
14: **end for**
15: **for each** $agent \in$ system **do**
16: finishTravel($agent, G_{\mathcal{L}}, A$)
17: **end for**
18: **end for**

### B. Parallelization of Flee

We implement two parallelization schemes: basic agent parallelization (AP) and agent-space parallelization (ASP).

*1) Agent parallelization (AP):* In AP scheme, we replicate location graph and distribute the agents evenly across the processes. When this is done, propagation of the whole system by one time step requires to aggregate agent totals twice [20]. MPI version of agent totals aggregation involves a single `MPI_Allgather` operation to sync all locations in bulk. On each process, it additionally requires a packing data before the collective call and an unpacking after.

*2) Agent-space parallelization (ASP):* To avoid a communication bottleneck for large and medium sized graphs, we developed an algorithm that distributes the responsibility of location updates across the different processes. It splits the locations list across all processes, and have each process update the locations assigned locally. This approach requires a single `MPI_Allgatherv` operation to synchronize the location scores across all processes.

### C. Complexity of the Algorithm

The computation of the P-Flee involves two time-consuming parts: (i) agent decision-making and (ii) updating location scores. The former dominates in the computational costs, while the latter defines the communication overhead.

Agent decision-making requires to evaluate up to $\min\left\{|V_{\mathcal{L}}|, \Delta^-(G_{\mathcal{L}})^A\right\}$ paths started at agent location $\ell$. Taking into account that agents are distributed evenly in both parallelization schemes, after summing up these contributions over all agents and $D$ days, we obtain the roof-line approximation for the computational complexity of P-Flee:

$$\mathcal{O}\left(\frac{D}{P}\left(N + \frac{D+1}{2}N_{new}\right)\min\left\{|V_{\mathcal{L}}|, \Delta^-(G_{\mathcal{L}})^A\right\}\right),$$

where $P$ is the number of utilized processes.

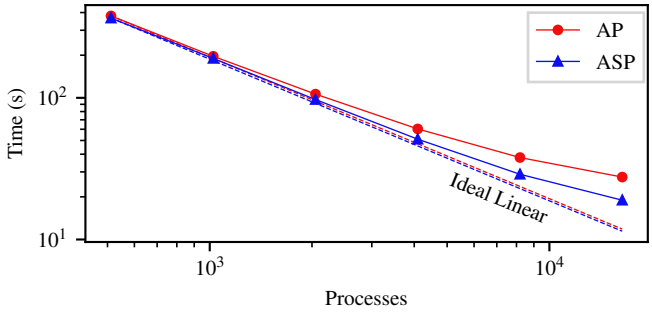



Figure 2. Comparison between the AP and ASP parallelization schemes of P-Flee, on a scenario with 100 million agents and a synthetic graph having 2500 locations and 4 routes per location, for 100 epochs.

## IV. Performance evaluation

### A. Experimental setup

To evaluate the P-Flee performance, we utilize four different testbeds and European supercomputers including two petascale (Eagle and Eagle2) and a new pre-exascale (Hawk) system. Table I shortly describes their characteristics. On all systems, we run P-Flee with Python dependencies *mpi4py 3.0.3*, *NumPy 1.19.0* and *SciPy 1.5.0*.

Table I
Experimental set up: hardware and software

| | **Hawk** | **Vulcan** | **Eagle** | **Eagle2** |
|---|---|---|---|---|
| Nodes | 5632 | 96 | 1087 | 1300 |
| CPU model | EPYC 7742 | Xeon 6248 | E5-2697V3 | Xeon 8268 |
| Cores/node | 2×64 | 2×20 | 2×14 | 2x24 |
| RAM/node | 256GB | 384GB | 256GB | 384GB |
| Interconnect | IB HDR200 | IB HDR100 | IB FDR56 | IB EDR |
| OS | CentOS 8.2 | CentOS 7.9 | CentOS 7.6 | CentOS 7.6 |
| File system | Lustre 2.12.5 | Lustre 2.12.6 | Lustre 1.8.0 | Lustre 2.12.6 |
| MPI | MPT 2.23 | OpenMPI4.0 | OpenMPI4.0 | OpenMPI4.0 |
| Interpreter | Python 3.8.3 | Python 3.6.6 | Python 3.7.3 | Python 3.7.3 |

### B. Results

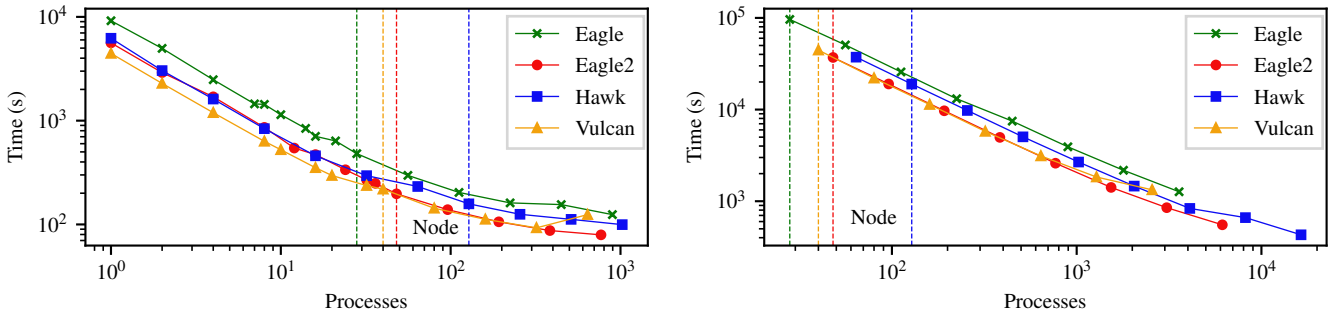
*1) Parallel Modes:* We first evaluate the agent-parallel (AP) and agent-space parallel (ASP) modes of P-Flee using a large simulation scenario with 100 million agents and a medium synthetic location graph containing 2500 locations and 4 routes per location, for 100 epochs on 512 up to 16384 cores (4-128 nodes) of Hawk. The results, shown in Fig. 2, indicate that the ASP mode, due to its parallelized location computations and its more efficient message packaging, significantly outperforms the AP mode. The benefit of ASP is particularly evident on high numbers of cores, while for up to 512 cores, the two modes produce similar execution times. Thus, we use the ASP mode in the consequent experiments.

*2) Simulating the case of South Sudan:* Subsequently, we evaluate the performance of P-Flee for the real world use case of South Sudanese Civil War, to assess the execution time and scalability of our codes on this and similar realistic scenarios. Fig. 3a demonstrates the execution time of Flee on a number of CPU cores ranging from 1 to 1024 on the aforementioned testbeds. Since each system features different cores with varying per-core performance, the execution time predictably differs from system to system. Nevertheless, we observe a similar trend in the scalability of Flee in all testbeds; thus, we can safely assume that the parallel performance of Flee is robust and independent of the underlying architecture details. It is worth annotating that the South Sudan case involves a small number of agents, starting from 800 thousand and ending with 2 million agents, therefore there is a limited granularity per process and limited potential performance gains from scaling this particular scenario on more cores. This results in the break of linear scalability for all systems after 32 processes. Additionally, the high variance observed on Vulcan at 768 processes is attributed to interference, since Vulcan is a small cluster (96 nodes) with a shared interconnect and filesystem, and the `MPI_Allgatherv` communication dominates P-Flee execution time in high core counts. Consequently, for the case of South Sudan and similar micro-scale scenarios, a moderate number of powerful cores is sufficient for parallel execution, which favors HPC systems with strong per-core performance like Eagle2 and Vulcan.

*3) Scalability study:* Finally, we evaluate the scalability of the parallel Flee code for a macro-scale scenario, for instance an international conflict or emergency situation where multiple adjacent countries are affected. To emulate a problem with more locations, we use 100 million agents and a larger synthetic graph having 10000 locations with 8 routes per location. In this scenario, the number of agents is kept fixed during the simulation ($N_{new} = 0$). We stopped our experiments on the benchmarks using 16384 cores due to the limitations for the job size on all testbeds. In Fig. 3b we present the results. For this scenario, the ASP version of P-Flee scales favourably to more than 16384 processes, as (i) the work per process increases with the number of agents and (ii) the overhead of setting up the simulation becomes negligible compared to the iterative phase of the simulation. However, the cost of setting up the simulation, as well as the costs of collective communication per time step which increase with the graph size, start to reduce the scalability of P-Flee on high process counts. This linear speedup break point varies for computer systems since it depends on the performance of the underlying file-system and interconnects. Nevertheless, the scalability results of the macro-scale simulation scenario indicate a preference for massively parallel, high bandwidth systems like Hawk.

## V. Conclusions and Future Work

In this work, we described the efforts to speedup a baseline serial Flee algorithm for simulating human migration by means of two parallelization schemes: basic agent parallelization (AP) and agent-space parallelization (ASP). We provided a roof-line computational complexity analysis of




(a) South Sudan conflict scenario for the full duration of 426 days (epochs)

(b) Synthetic macro-scale scenario for 100 epochs

Figure 3. Evaluating the execution time of P-Flee on four machines. The vertical dashed lines show the single-node core count for each machine.

these schemes and evaluated their performance on four state-of-the-art HPC platforms. These experiments demonstrated the improvement derived from the ASP mode with the high scalability of our codes on all target testbeds, and provided some insights for choosing the HPC system depending on the simulation scenario.

With the revision of the decision-making rulesets and of the location and path objects, P-Flee generalizes naturally to the wider scope of migration questions, including forecasting the longer-term human migration and the migration of other species. Moreover, the parallelization schemes of P-Flee can be re-engineered for some other ABSS applications with frequent agent movement and much lower impact of direct communications between agents, such as modelling of trade, transportation, and movement of goods. In future work, we intend to study these potential applications of P-Flee.

## Acknowledgements

This work has been supported by the HiDALGO project and has been partly funded by the European Commission's (EC) ICT activity of the H2020 Programme under grant agreement number: 824115. This paper expresses the opinions of the authors and not necessarily of the EC.

## References

[1] UNHCR, "Figures at a Glance," Available at: https://www.unhcr.org/figures-at-a-glance.html, 2020.

[2] D. Suleimenova, D. Bell, and D. Groen, "A generalized simulation development approach for predicting refugee destinations," *Sci. Rep.*, vol. 7, no. 1, pp. 1–13, 2017.

[3] S. Edwards, "Computational tools in predicting and assessing forced migration," *J. of Refugee Studies*, vol. 21, no. 3, pp. 347–359, 2008.

[4] G. A. Hébert, L. Perez, and S. Harati, "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*. Springer, 2018, pp. 45–58.

[5] A. J. Collins and E. Frydenlund, "Agent-based modeling and strategic group formation: a refugee case study," in *WSC*. 2016, pp. 1289–1300.

[6] A. Hattle, K. S. Yang, and S. Zeng, "Modeling the Syrian refugee crisis with agents and systems." *UMAP Journal*, vol. 37, no. 2, 2016.

[7] N. Blandin, C. Colglazier, J. O'Hare, and P. Brenner, "Parallel Python for agent-based modeling at a global scale," in *Proc. of International Conference of The Computational Social Science Society of the Americas*, 2017, pp. 1–7.

[8] N. Collier and M. North, "Parallel agent-based simulation with Repast for high performance computing," *SIMULATION*, vol. 89, no. 10, pp. 1215–1235, 2013.

[9] E. Tatara, N. Collier, J. Ozik, and C. Macal, "Endogenous social networks from large-scale agent-based models," in *IPDPSW, ParSocial*, 5 2017.

[10] G. Cordasco, V. Scarano, and C. Spagnuolo, "Distributed MASON: A scalable distributed multi-agent simulation environment," *Simul. Model. Pract. Th.*, vol. 89, pp. 15–34, 2018.

[11] C. L. Barrett, K. R. Bisset, S. G. Eubank, X. Feng, and M. V. Marathe, "EpiSimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks," in *SC*, 11 2008.

[12] D.L. Chao, M.E. Halloran, V.J. Obenchain, and I.M. Longini, "Flute, a publicly available stochastic influenza epidemic simulation model," *PLoS Comp. Bio.*, vol.6, no.1, 2010.

[13] K. S. Perumalla and S. K. Seal, "Discrete event modeling and massively parallel execution of epidemic outbreak phenomena," *SIMULATION*, vol. 88, no. 7, pp. 768–783, 2012.

[14] J. Yeom, A. Bhatele, K. Bisset, E. Bohm, A. Gupta, L. Kale, M. Marathe, D. Nikolopoulos, M. Schulz, and L. Wesolowski, "Overcoming the scalability challenges of epidemic simulations on Blue Waters," in *IPDPS*, 2014, pp. 755–764.

[15] Y. Wu, W. Cai, Z. Li, W. J. Tan, and X. Hou, "Efficient parallel simulation over large-scale social contact networks," *ACM Trans. Model. Comput. Simul.*, vol. 29, no. 2, Apr. 2019.

[16] T. G. Schmitt, S. Kumar, K. E. Stecke, F. W. Glover, and M. A. Ehlen, "Mitigating disruptions in a multi-echelon supply chain using adaptive ordering," *Omega*, vol. 68, pp. 185 – 198, 2017.

[17] D. Hristova, M. J. Williams, M. Musolesi, P. Panzarasa, and C. Mascolo, "Measuring urban social diversity using interconnected geo-social networks," in *WWW*. 2016.

[18] K. Nagel and M. Rickert, "Parallel implementation of the TRANSIMS micro-simulation," *Parallel Computing*, vol. 27, no. 12, pp. 1611 – 1639, 2001.

[19] S. Gogolenko, "Large scale agent-based social simulations with high resolution raster inputs in distributed HPC environments," in *Sustained Simulation Performance 2018 and 2019*, Springer, 2020, pp. 205–214.

[20] D. Groen, "Development of a multiscale simulation approach for forced migration," in *ICCS*. Springer, 2018, pp. 869–875.