Parallel Simulation of Community-Wide Information Spreading in Online Social Networks

Sergey Kesarev, Oksana Severiukhina and Klavdiya Bochenina

ITMO University, Saint Petersburg, Russia,
{kesarevs, oseveryukhina, k.bochenina}@gmail.com

Abstract. Models of information spread in online social networks (OSNs) are in high demand these days. Most of them consider peer-to-peer interaction on a predefined topology of friend network. However, in particular types of OSNs the largest information cascades are observed during the community-user interaction when communities play the role of superspreaders for their audience. In the paper, we consider the problem of the parallel simulation of community-wide information spreading in large-scale (up to dozens of millions of nodes) networks. The efficiency of parallel algorithm is studied for synthetic and real-world social networks from VK.com using the Lomonosov supercomputer (Moscow State University, Russian Federation).

Keywords: Parallel simulation · Model of information spread · Online social networks

1 Introduction

Wide spread of online social networks (OSNs) gave rise to a variety of models aimed at simulating and forecasting processes of information assimilation and transmission between users. While first models of this type were purely abstract (e.g. linear threshold models of opinion dynamics operating on syntetic graphs like Erdos-Renyi or Barabasi-Albert), current availability of large amounts of OSN data allows to create more realistic, data-driven models of information cascades. These models avoid the assumption of the homogeneity of nodes that was intrinsic to earlier ones; instead, these models are aimed to reproduce aggregated dynamics of information spread from reactions of diverse individuals acting in frames of a given network topology. The main difference then is that friendship graph and parameters of users' behavior are extracted directly from the observed data.

Data-driven approach not only refines the basic models but also states new problems arising from the nature of data observed. In this paper, we consider the example of such a problem related to the existence of 'super-hubs' (communities) in OSN which numbers of subscribers are orders of magnitude larger than for individuals (ordinary users). For example, a median number of subscribers for an individual in Russian OSN vk.com is equal to 180 while communities can have size of several hunderds thousands and even millions subscribers. Thus,

simulating community-wise information spread requires consideration of large-scale networks. Together with the rapidity of informational processes in OSNs this leads to the necessity of applying parallel algorithms to obtain results of a forecast in time.

In this study, we present the parallel algorithm of the community-wise information spread supporting the flow of heterogeneous news (with account of daily rhythms) through the network of community subscribers, friends of subscribers etc. Presented algorithm (Section 3), on the one hand, is general in a sense that it supports embedding of arbitrary models of activities and reactions of OSN entities, but, on the other hand, is tuned to the specific properties of OSN. Section 4 presents the results of experiments on scalability of the algorithm for single- and multi-community cases, for real-world and synthetic test cases.

2 Related Work

The modeling of the process of information dissemination in social networks has a wide range of applications: the study of factors that affect the process of information dissemination, the prediction of reaction, the maximization of influence, rumor controlling, the evaluation of public opinion, etc.

In the task of modeling processes on large networks, researchers and developers are faced with a large number of problems associated with the heterogeneity of vertex types, the amount of data transferred, and the distribution of vertices by processes. Load balancing algorithms on graphs can be divided into several groups depending on the problem under consideration: (i) type of process (spreading information, epidemiological processes), (ii) network topology (real world network or artificial like small-world, scale-free), (iii) possibility of generalization (algorithms for specific tasks or multi-purpose systems for parallel graph processing).

Systems of the first class take into account features of the simulated process. The most common problem is the modeling of epidemiological processes. Examples of systems of this class are EpiFast [1], EpiSimdemics [2] and Indemics [3]. EpiFast has a master-slave computation model. Experiments show that it shows scalability to 224 processes for approximately 16 million of entities. However, major drawback which has an impact on scalability is the arbitrary assignment of vertices to processes containing approximately the same number of edges. Next example is EpiSimdemics, which uses semantics of disease evolution and disease propagation in large networks (up to 100 million) and considers geospatial constraints. The above mentioned approaches use the MPI standard for parallelisation. Another solution that takes into account features of the process under consideration is Indemics. It has an interface for work and functionality for stop the simulation at any point, find out the state of the simulated system and add additional interventions.

Modeling and simulation of social networks focuses more on other processes like reproducing complex social phenomena, such as spreading of information and its impact on others, maximization influence, identification of opinion lead-

ers and what-if tasks. Hou [4] proposed framework named SUPE-Net for parallel discrete event simulation. It has utilities for social network generation, algorithms (PageRank and the SIR model in study), which show the scalability and effectiveness of this framework. To distribute the vertices of the network by processes, the authors use the algorithm CommPar [5], which has a community-based multilevel graph partition strategy. However, for this algorithm it is required to know the number of available processes. Nevertheless, it efficiently reduces the overhead of communications between processors. Another example is using parallel algorithms for community detection for networks, for example, work [6] can process billions of edges in short time (50M edges per second for the fastest algorithm).

Thus, solutions of this class most often provide scalability of parallel algorithms with additional constraints on the interaction of vertices, which significantly reduces the communication complexity of modeling.

In the second approach, modeling techniques are considered on networks with a special topology. For example, authors [7] consider a stochastic Kronecker graph, which allows to reproduce real-world networks and keep their important topological properties. Experiments were carried out for a sparse graph with a size up to one billion nodes.

Last group of systems are general systems for parallel processing of graphs. They use different computing models to implement parallelization. Pregel [8] system uses Bulk Synchronous Parallel model. Apache Giraph [9] is based on previous mentioned system Pregel, however, it has several improvements like master computation, sharded aggregators, edge-oriented input. Another approach is using asynchronous model in a shared-memory setting, for example, GraphLab [10].

In work [11], Aydın Buluc reviewed various algorithms for partitioning graphs into parts: global algorithms for small graphs or as local search in multilevel algorithms, iterative improvement heuristics which consistently improve the solution, multilevel graph partitioning and evolutionary methods. Besides this, there is solution for streaming graph partitioning [12], which able to compute an approximately balanced partitioning of graph's vertex. This solution uses degree-based criteria and reaches the results for single pass over the input data.

If there is an uneven distribution of vertices on different processes, there is an uneven computing load between the nodes. However, if different processes have a large number of common edges, then much time is required to transfer data between them. Thus, in order to reduce the running time of the program, it is necessary to distribute the vertices between processes in such a way that the most related components are processed in one process.

The existing algorithms do not take into account the presence of superspreaders that arise during modeling processes on networks. So for the task of spreading information from, superspreaders are communities, which have edges with a large number of subscribers, and individual users with a large number of subscribers.

3 Method

3.1 Model Description

In order to reproduce information spreading processes in cyberspace more naturally it is necessary to consider a lot of details such as types of entities, their possible actions and interactions between them. The model used for our experiments is described in detail in our previous paper [13]. Below there is a brief description of it.

The model consists of three main entities: informational messages (IMs), communities and users. Networks for information spreading include communities and users as vertices and relations between them (subscriptions or friendship relations) as edges. IM represents the post, which can be transferred between vertices.

The behavior of presented entities is defined by three internal models: model of IM's generation, model of activity and model of reaction. First model determines the time for the creation of new messages. Each message has following characteristics: topic, publication time, virality coefficient. Model of activity defines the status of each agent: active or inactive. The last internal model is responsible for the result of the user's interaction with messages: inaction, approval (like), the generation of a text message (comment), participation in the dissemination of information (share). Each message has counters that reflect the number of users' responses to the message at the current time.

The presented approach allows to use various independent internal models to adjust the necessary process for modeling. In addition, each model can be specified in the appropriate input file. For the generative model input can contain the distribution of the probability for generating messages depending on the day of the week and time of day. In the presented model, each user can have his own parameters for responding to messages, thus providing the heterogeneity of agents.

Thus, the presented model allows you to simulate various processes and configure model parameters, for e.g. using processed data from social networks.

3.2 Parallelization

The efficiency of the parallel simulation of the information spreading on a social network depends a lot on the uniformity of computational and communication load for different workers. In our previous work, a similar problem was already examined for the task of epidemiological process modeling (SIRS) on stochastic Kronecker graphs [7]. That study has shown that Master-Slave parallelizing approach shows the best parallel efficiency. A reincarnation of this approach is used in the current study to handle parallelization challenges of a social network graph.

Detailed algorithms for Slave and Master processes are presented in Algorithm 1 and Algorithm 2 respectively. The simulation is discrete and operates

according to 3 internal models that were mentioned in the previous section: generative model G_m, activity model A_m and reaction model R_m. The algorithm is implemented in C++ using MPI standard for message exchange.

Masters and Slaves are two types of computational nodes, differing by their functionality. Slave nodes host subsets of a social graph and perform iterative updates of the system state according to internal models. The Master node is responsible for fast and non-redundant data forwarding between subnetworks. The responsibility to generate news and store user reaction statistics for them also rests with Master node. Primary reasons for choosing Master node instead of Slave nodes here are to circumvent the need to develop the synchronous news generating engine on each Slave node and to reduce synchronization time between Slave processes.

The goal of the simulation is to represent the process of the information spread step-by-step, not omitting any of transition states. This implies the necessity to maintain the relationship between each user and each piece of news for each moment of time. On the other side, memory usage in the process of the simulation should be as small as possible. Having this limitation in mind, we propose the news-oriented storing system. That is, for each piece of news on each Slave node there are defined three boolean masks that describe the state of publication regarding users on this node:

- potential viewers — stores 1 in positions corresponding to users who haven't yet seen this publication, but have it included in their news feed; for example, for some post that is published a moment ago, all subscribers of the community of this post are potential viewers of it (if they were not reading their news feed at the very moment of publication);
- spreaders — users who decided to share this piece of news with all their subscribers;
- viewers — users who have already seen this publication.

On each iteration, Slave process receives the list of generated news from the Master node (line 2 in Algorithm 1). Then each publication is processed in the chronological order, from newest to oldest posts (line 3).

In the first phase, the list of potential viewers of this publication is examined (line 4): we attempt to show the publication to the user and gather his feedback about it. If the user is not active according to the activity model A_m, the algorithm moves on to the next potential viewer, and this one stays in the list to be checked on the next iteration (lines 6). If the user is active, the flow continues. Algorithm checks according to reaction model R_m whether the user likes the publication or wants to comment it (lines 8-10), and if he does, the algorithm increases corresponding counters for the publication. The next check is whether the user wants to share the publication (lines 12), and if he does, he is added to the bit mask of spreaders of this publication. When the reaction is gathered, the user is deleted from the bit mask of potential viewers (line 13) and is added to the list of those who saw this publication (line 14).

The second phase of processing each publication is processing of the list of spreaders (which was constructed while examining potential viewers). Each

Input : Activity model A_m, reaction model R_m, news feed in the chronological order N, a number of iterations T, M — master node for worker w, `all_masters` — the list of all Master processes

```
1  for t from 1 to T :
2    syncNews(M);
3    foreach n ∈ N :
4      foreach p ∈ potentialViewers(n) :
5        if not A_m.isActive(p, t) :
6          continue
7        if R_m.isLike(n, p, t) :
8          addLike(n, p, t)
9        if R_m.isComment(n, p, t) :
10         addComment(n, p, t)
11       if R_m.isRepost(n, p, t) :
12         addRepost(n, p, t)
13       deletePotentialViewer(n, p);
14       addViewer(n, p);
15     foreach s ∈ spreaders(n) :
16       for e ∈ s.edges :
17         if dest(e) ≠ w :
18           send_pools[dest(e)] ← (n, e)
19         else:
20           if not isViewed(n, e) :
21             addPotentialViewer(n, e)
22     deleteAllSpreaders(n);
23     sendInfoMessageToMaster(M);
24     sendPoolsToMaster(M);
25     for each (n, e) in
         recv_pools(all_masters) :
26       if not isViewed(n, e) :
27         addPotentialViewer(n, e)
```

Algorithm 1: Parallel simulation scheme for the information spreading process(for a Slave worker w)

subscriber of each spreader from the list is offered this publication, i.e., is added to the list of potential viewers of this publication (lines 15-21). If a particular subscriber is not hosted on the current computational node, information about him and this publication is added to the message that will be sent to the Master node (lines 18). If a subscriber is on the current node, then, if he has not already seen the publication, he is added to the list of potential viewers of this publication (line 21). By this moment the publication is set to be offered to all subscribers, and the list of spreaders can be deleted (line 22).

When the algorithm processed in the described way all publications, it is time to share results with other parts of the simulation system. After processing spreaders of each news we obtained a set of pairs (n, e) of a user e from some other node who became a potential viewer of a post n (see line 21). First of all, the Slave node sends an informational message to Master node (line 23). This message shows how many changes ((n, e) pairs) should be sent to the each other Slave node in the system. Then a message containing all pairs grouped by hosting computational nodes is sent (line 24). When the Master node has processed this message along with others (processing details are thoroughly described below), it sends back a message with the list of changes for the current Slave node. This message contains pairs (n, e) obtained in the process of information spreading on other Slave nodes. Each pair is processed as usual: if e has not yet seen n, then e is added to the list of potential viewers of n. Only after processing all these messages a node can move on to the next iteration.

Input : $i*$ – an index of a master process, G_m – generative model, A_m – activity model, N – news feed in the chronological order, T – number of iterations, `all_leafs` — the list of all Slave nodes, `own_leafs` — the list of Slaves that obey directly to this Master node

```
1  for i from 1 to T :
2    if i* = 0 :
3      generteNews(t);
4      syncNewsWithLeafs(all_leafs);
5    foreach l ∈ own_leafs :
6      infos ← recvInitInfo(l);
7    consolidatedPool ← allocateMemory(infos);
8    foreach l ∈ own_leafs :
9      nonBlockingRecv(l, consolidatedPool, infos);
10   waitForAllRecvs();
11   types ← createMpiDatatypes(infos);
12   dispatchPool(all_leafs, types, consolidatedPool);
```

Algorithm 2: Parallel simulation scheme for the information spreading process(for a Master worker w)

The primary task of a Master process on each iteration is to synchronize states of its Slaves. Generally speaking, there may be several Master processes, and each Master node cares only for a subset of Slaves (which is called `own_leafs` in the Algorithm 2). However, if there are several Master nodes, we still have to choose the only one which will be responsible for news generation. For this selected Master node each iteration starts with generating news according to generative model G_m and syncing them with all Slave nodes (lines 2-4). The next step is receiving informational messages from Slave nodes (line 5). Master node allocates memory for the consolidated pool according to the received informational message (line 7). When the memory is allocated, Master node receives detailed information from its Slaves to this memory (lines 8-9).

The only thing left is to send to each Slave node all changes that are related to this node. For each message that will be sent to the Slave node, we are creating an MPI datatype according to informational messages received before (line 11). This approach allows avoiding additional memory allocation. Then these messages are sent to Slave nodes and are processed by them (line 12).

4 Experiments

4.1 VK Dataset Description

The first dataset is the example crawled from the popular Russian social network VK.com (hereafter it will be called VK dataset). The crawled graph consists of one big charity community (294,345 members), all its members and all their subscribers. The graph contains 33,768,036 vertices (i.e. accounts in social network), and only 227,473 of them have at least one outgoing edge (i.e. at least

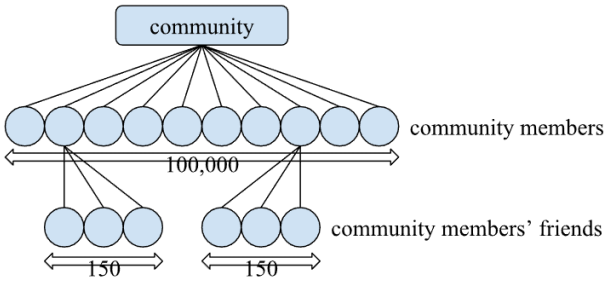

Fig. 1. Artificial dataset scheme

one friend). Note that this is not because other 33 millions accounts do not have friends, but because the fact of having friends other than community members is not recorded. That means there are at least 66,872 community members that do not have subscribers at all and that there are 5,262,115 nodes who are friends to two or more community members at the same time. A more thorough description is given in our recent work [13].

4.2 Artificial Dataset Description

Characteristics of the VK dataset allow to conclude that an n-ary tree might be a good rough approximation of VK graph. Let us think about the community as a tree root which has a big pre-specified number of child nodes: these will be community members. Each of these community member nodes has a fixed pre-specified number of child nodes, which will represent their friends. These friend sets are not intersected (i.e. there are no nodes that are subscribed to more than one community member). A forest of such trees was used as an artificial dataset to test some hypotheses about algorithm efficiency with respect to the number of communities. The graph scheme is presented in Figure 1.

4.3 Results on the Natural Data

All experiments were performed using Lomonosov supercomputer (Moscow State University, Russian Federation) which has 4,096 cluster nodes with Intel Xeon X5570 2.93 GHz processor and 12 GB of RAM per node. Each cluster node can hold up to 8 processes.

The algorithm from the previous section was applied at first to the VK dataset and run for 3,000 iterations (500 hours of model time) in several parallelization settings: a sequential version and parallel version on 3, 8, 16, 32 or 64 processes.

Simulation dynamics are presented in Fig. 2. Computational load does not significantly increase in time despite the growing number of publications in the system, even though publication processing is the main source of computational load in the system.

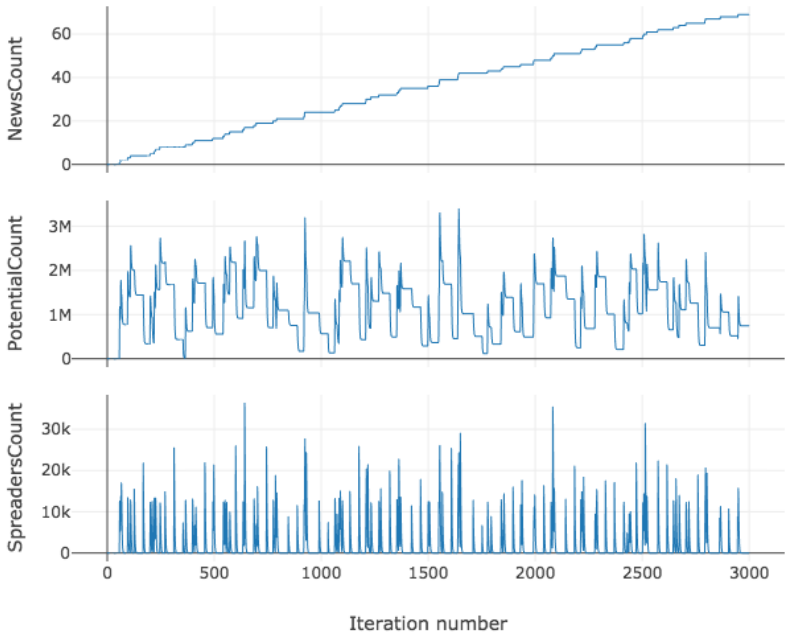

Fig. 2. Simulation dynamics

Parallelization allows us to significantly reduce simulation time (which is shown in the Fig. 3, exact timing is shown in Table 1). More specifically, modeling of 500 hours of the network evolution can be done in several minutes. This means the algorithm may be used in the day-to-day operative forecasting of the information flow.

However, even with sufficient computation-to-communication ratio (around $^2/_3$ of iteration time is spent on computations, mostly processing potential viewers, and $^1/_3$ on communication, see Fig. 4) the parallel efficiency of the algorithm is low. This is because the computational load is not increasing during simulation.

Table 1. Simulation time by the number of processes on VK dataset

Number of processes	Simulation time, seconds
1	19820.20
3	13187.40
8	10180.00
16	6959.38
32	3513.66
64	1793.57

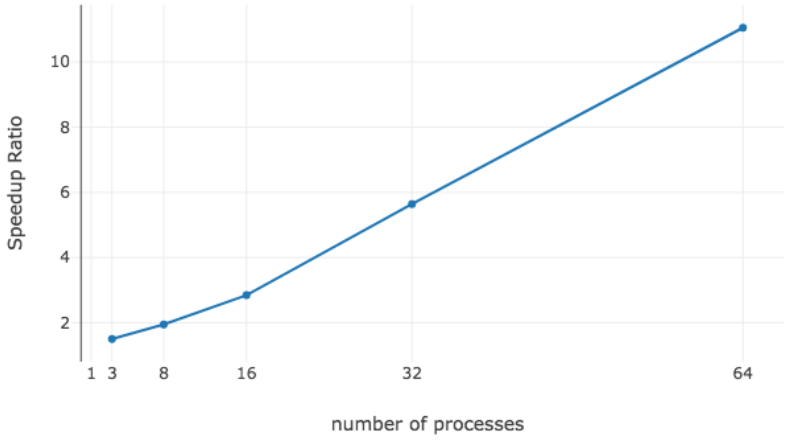

Fig. 3. Speedup ratio on the VK dataset for the different number of processes

Computational load increases when information cascades become deeper and more intensive. Intensity could be increased either by increasing probabilities of the news generating model or by increasing the number of communities in the dataset. Increasing probabilities, however, makes the model less plausible. Therefore for deeper understanding of the algorithm properties, it was run on an artificial network with several communities. This network was described in details in the section 4.2.

4.4 Results on Artificial Data

Table 2. Simulation time by the number of processes on artificial dataset, seconds

Processes:	1	3	8	16	32	64	128
Communities							
5	28420.10	16017.30	8001.07	5391.60	2893.30	1570.42	978.86
4	20393.20	11523.10	7263.05	4387.40	2246.57	1212.05	759.22
3	14509.60	8239.09	5966.98	3485.21	1683.66	896.08	634.81

The goal of the experiment with artificial data was to understand the relation between parallel efficiency and the number of communities in the network.

Parallel efficiency for different numbers of processes for 3,000 iterations is presented in the Figure 5. Exact timings are available in the Table 2. It is clearly seen there that the parallel efficiency of the algorithm increases with the number of communities in the graph. Almost exponentially decreasing simulation time

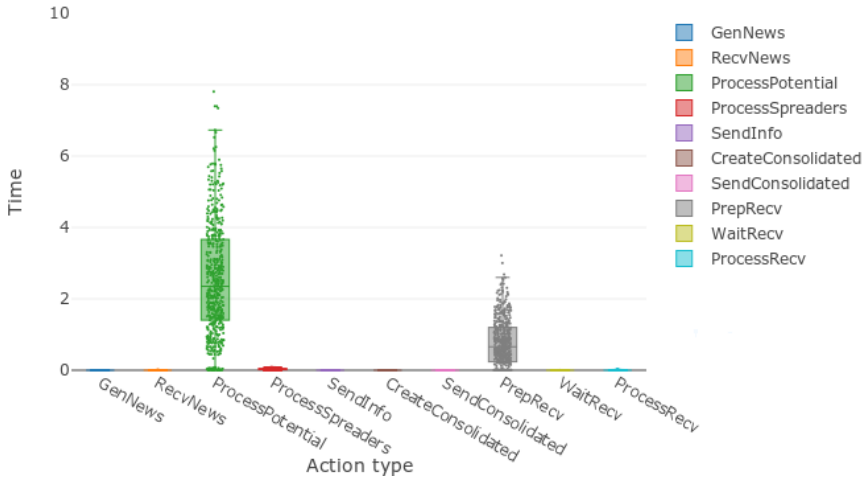

Fig. 4. Time spent on different stages of simulation (8 processes)

when the number of processes increases from 16 to 32, 64 and 128 for any number of communities can mark the insufficient computational load in the simulation.

That is why future research directions are towards modeling a multi-community landscape with mixing audience, creating specialized load balancing algorithms for such communities and involving Master processes in the active computations.

5 Conclusion

We presented an algorithm for parallel simulation of community-oriented information spread which allows easy modification of behavioral patterns of users and communities via probabilistic models. The algorithm was applied to the subset of the VK.com, the largest Russian social network. The subset contained all members of a big charity community and all their friends. Experimental results show the applicability of this algorithm for day-to-day modeling of planned information cascades. However, experiments also revealed insufficient computational load provided by this dataset. For the detailed study of algorithm properties it was applied to the artificial dataset of similar structure that included several communities. The increasing number of communities increases the number of news and therefore rises up computational load, allowing the proposed method to finally prove its applicability.

Acknowledgements

This research was supported by The Russian Scientific Foundation, Agreement #14-21-00137-П (02.05.2017). The research was carried out using the equip-

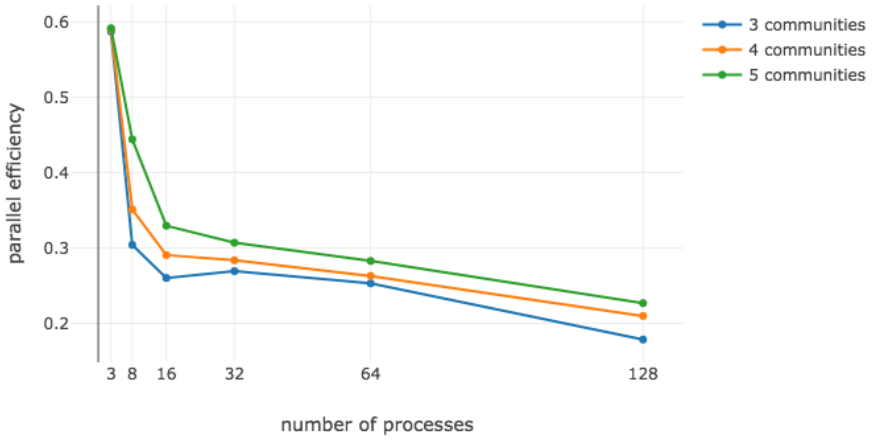

Fig. 5. Parallel efficiency on artificial data

ment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. Bisset, K.R., Chen, J., Feng, X., Kumar, V.S.A., Marathe, M. V: EpiFast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems. In: Proceedings of the 23rd International Conference on Supercomputing. pp. 430–439 (2009)
2. Barrett C, Bisset, K.R., Eubank Stephen G, Feng X, Marathe, M.: EpiSimdemics: An efficient and scalable framework for simulating the spread of infectious disease on large social networks. Int. Conf. High Perform. Comput. Networking, Storage Anal. (2008). doi:10.1145/1413370.1413408
3. Bisset, K.R., Chen, J., Deodhar, S., Feng, X., Ma, Y., Marathe, M. V: Indemics: An Interactive High-Performance Computing Framework for Data Intensive Epidemic Modeling. ACM Trans. Model. Comput. Simul. 24, 1–32 (2014). doi:10.1145/2501602
4. Hou, B., Yao, Y., Wang, B., Liao, D.: Modeling and simulation of large-scale social networks using parallel discrete event simulation. Simulation. 89, 1173–1183 (2013). doi:10.1177/0037549713495752
5. Hou, B., Yao, Y.: COMMPAR: A community-based model partitioning approach for large-scale networked social dynamics simulation. In: Proceedings - IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT. pp. 7–13. IEEE (2010)
6. Staudt, C.L., Meyerhenke, H.: Engineering Parallel Algorithms for Community Detection in Massive Networks. IEEE Trans. Parallel Distrib. Syst. 27, 171–184 (2016). doi:10.1109/TPDS.2015.2390633
7. Bochenina, K., Kesarev, S., Boukhanovsky, A.: Scalable parallel simulation of dynamical processes on large stochastic Kronecker graphs. Futur. Gener. Comput. Syst. 78, 502–515 (2017). doi:10.1016/j.future.2017.07.021

8. Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the 28th ACM symposium on Principles of distributed computing - PODC '09. p. 6 (2009)
9. Apache Giraph, http://giraph.apache.org/
10. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., Hellerstein, J.M.: Distributed GraphLab: A Framework for Machine Learning in the Cloud. 716–727 (2012). doi:10.14778/2212351.2212354
11. Buluc, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent Advances in Graph Partitioning. CoRR. abs/1311.3, (2013). doi:10.1007/978-3-319-49487-6_4
12. Tsourakakis, C.E.: Streaming Graph Partitioning in the Planted Partition Model. (2014). doi:10.1145/2817946.2817950
13. Severiukhina, O., Bochenina, K., Kesarev, S., Boukhanovsky, A.: Parallel data-driven modeling of information spread in social networks. Lecture Notes in Computer Science, IET - 2018, Vol. 130, pp. 248-260