A Fine-grained Parallel Approach for one Logical Process on Multi-core Machines

Jiawei Fei
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha
China
feijiawei11@nudt.edu.cn

Yiping Yao
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha
China
ypyao@nudt.edu.cn

Jing Luan
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha
China
luanjing@mail.ustc.edu.cn

Lufan Li
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha
China
lilufan0315@gmail.com

Yingqian Bao
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha
China
baoyingqian15@nudt.edu.cn

ABSTRACT

Currently, the time management algorithms applied in various parallel discrete event simulation (PDES) engines take the logical process as the smallest parallel unit which corresponds to a physical process and represents a sequential simulation. Before the simulation system is running, all the entities are distributed to each logical process. The parallelism of the simulation system depends on the parallelism between logical processes. The performance of this parallel approach is greatly affected by the entity distribution scheme. For example, when the computing hotspots are assigned to one LP, the performance of the parallel simulation is almost the same as that of the serial simulation. And it is difficult to find a suitable entity distribution scheme when it comes to simulation with hotspots which migrate between LPs dynamically. Besides, we always use as many LPs as possible to improve parallelism, but this also brings greater communication and synchronization overhead. Focusing on these drawbacks of current simulation engines, we propose an approach which supports fine grain parallelism in one LP. Based on the traditional time management algorithms, our approach can process events of different entities in one LP in parallel with multi threads.

At the same time, we propose a multi-level GVT management method to achieve synchronization in thread level and LP level efficiently. Finally, to evaluate the performance of our approach, we use Pholds benchmarks including star structure Pholds and ring structure Pholds to test our optimized simulation engine. Results show that, in either case, the optimized simulation engine shows higher performance. Especially when the load between LPs is unbalanced, our method can achieve more than 3 times performance improvement compared with original simulation engine.

SIMUTOOLS '17, September 11–13, 2017, Hong Kong, China

ACM ISBN 978-1-4503-6388-4/17/09…$15.00
https://doi.org/10.1145/3173519.3173541

CCS CONCEPTS

• **Computing methodologies** → **Simulation tools**; *Massively parallel and high-performance simulations*

KEYWORDS

PDES, Logical Process, Fine-grained Parallel, Multi-core.

1 INTRODUCTION

Parallel discrete event simulation engines gain great performance improvement by using multi machines of clusters in parallel. However, the outstanding computing ability of the multi-core machine is always ignored. Taking full advantage of the multi-core machine to speed up simulation engines has become a research hotspot.

As we all know, the time management algorithm is the key technology to implement event parallel processing for simulation engines. Logical processes (LPs) are the minimal parallel units in time management algorithms. A logical process represents a

sequential simulation which processes events of one entity or multi entities serially. Before executing time management algorithms, the simulation engines need to distribute simulation entities to LPs. Different LPs execute in parallel and each LP processes events of entities distributed to it serially. Time management algorithms bring great performance improvement with parallelization. However, some factors limit the parallel performance especially when it comes to multi-core machines. Firstly, the distribution of entities has great influence on the quality of load balancing. When the computing hotspot is too concentrated, the parallel performance will decrease a lot. Many load balancing methods have been proposed focusing on the load balancing problem but do not show ideal effects due to the randomness of events generated by PDES applications. Another factor limiting the performance is the number of LPs. To maximizing parallel performance, the number of LPs is always set to the core number of multi-core machines. However, too many LPs will cause large memory consumption and lots of extra communication overheads.

Concentrating on the factors limiting parallel performance, many researchers begin to pay attention to the multi-core architecture. These optimized methods mainly gain the performance by modifying the implement of algorithm. For example, some simulators like Threaded Warp [1] and ROSS-MT [2] take advantage of the shared memory of multi-core architecture to optimize the communication among LPs in one node. However, using shared memory to optimize communication can only reduce the cost of one communication process. The communication and synchronization cost caused by the increase of the LP number cannot be reduced. And we find that increasing parallelism and reducing communication and synchronization cost are contradictory in current time management algorithms. In addition, load balancing problem is also concerned by researchers. Many static and dynamic load balancing methods have been proposed and most of them are based on the performance metric. However, the effects of these methods are not ideal because the tasks generated by PDES applications are random and the computing hotspots will migrate between LPs during execution.

This paper focuses on the demands described above and proposes an approach to improve the performance of simulation engines by processing events in one LP with multi-thread technology. With our approach, the load balancing problem can be solved because even if computing hotspots are in one LP, the computing tasks of one LP will be assigned on multi cores. What's more, our approach can reduce the number of LPs without the loss of parallelism which can cut down the cost of communication and synchronization among different LPs and minimize memory usage.

The paper is organized as follows, Section 2 gives the related works on PDES engines and multi-core technologies used in simulation. Section 3 introduced the fine grain parallelism in one LP. Section 4 evaluate the optimized approach using Pholds benchmark. At last, the conclusion is presented in Section 5.

2 RELATED WORKS

Multi-core platform has been the focus of researchers for its advantages including shared memory and low cost for parallelization. In the field of PDES, multi-core simulation optimization has also made great progress. These achievements are applied in many famous simulation engines. Threaded Warped is a multi-core optimization version of the simulation engine Warped which was designed by Miller, R.J. [1]. It focuses on the time management optimistic algorithm Time Warp and adopts the master-slave working mode to manage message queues. In Threaded Warped, even if both the sender and receiver object are on the same node, the event message still must pass through the message queue. These events cannot be inserted directly into the receiving object's input queue. And GVT computation requires extra cost and all threads must be suspended in the synchronous operation. ROSS-MT is another simulation engine which supports multi-core platform [2]. It is based on ROSS which can only run in multi-process mode. Wang, J. et al. optimized the message passing mechanism of ROSS with shared memory to reduce communication cost. However, ROSS-MT cannot run in mixed mode with multi-thread and multi-process modes which means it cannot use multi machines in multi-thread mode. A relatively new research about accelerating simulators on multi-core machine is conducted by Yang, C. et al [3, 4]. They proposed a parallel simulators Ivy under multi-core environment for the simulation of the variable structures system which exhibits exchanges both at structural and behavior levels. To solve the load balancing problem, Yang, C. et al adopted a dynamic load balancing method which can migrate models among cores with very low cost and support dynamic core allocation on demand. Ivy only proposed the solution for multi-core environments without consideration of clusters with multi machines. Besides, some other simulators also support multi-core platforms like GTW (Georgia Tech Time Warp) [5] which developed by Fujimoto team and NTW-MT [6] which is a multi-threaded simulator for the simulations of reaction diffusion developed by Lin, Z. et al.

Load balancing technologies are important technologies to improve the efficiency of hardware. They can be divided into two categories, static and dynamic methods on the basis of the time to adjust load. Especially for the PDES applications, randomness of load generation brings great difficulty to load balance. At present, the load balancing research achievements of PDES applications are remarkable. Most of these researches are based on the performance metrics. Many researches intend to propose a metric that can evaluate performance well. For example, the metric for detecting load imbalances called simulation advance rate was proposed by Glazer and Tropper [7] and the metric of computation load depended on the number of events processed and the effective time advance are proposed by Peschlow [8] in 2017. Recently, Yang, C. et al. [4] proposed a new performance metric based on simulation advance rate by considering the processing time of events. These metrics serve as the basis for load distribution in current load balancing methods but they can only evaluate performance approximately by some measurable indicators like running time and event number. Some other dynamic load balancing strategies do not apply these metrics, they migrate the load waiting for process to idle computing unit at runtime. Focusing on these strategies, Ahn, T. H. [9] introduced and evaluated them detailedly in his research. The first is Most-Dividing (MD) algorithm based

on the central redistribution work of Powley [10] and Hillis [11], in which the most overloaded processor needs to send out half of its remaining jobs to other idle processors. The second is All-Redistribution (AR) algorithm which is similar to MD algorithm but the load distribution work is carried out by the master. The third is Random-Polling (RP) algorithm [12] which is a receiver-initiated decentralized load balancing algorithm. The last is Neighbor-Redistribution (NR) algorithm which only supports the load migration between neighbors. These dynamic load balancing strategies bring much extra migration cost.

3 Fine Grain Parallelism in one LP

3.1 Introduction of PDES Engine SUPE

Time management algorithms can be divided into two categories including conservative algorithm and optimistic algorithm. We mainly discuss conservative algorithm because the rollback operation in optimistic algorithm is usually hard and costly. In time management algorithms, physical processes are modeled by logical processes (LPs) and the interactions between physical processes are modeled by exchanging messages between LPs. So PDES application is modeled by a collection of collaborative LPs. When building PDES system, we need to design the simulation entities and events. In the simulation engine level, the relationship between the simulation entities and LP is many to one what means that each LP will manage multi entities. If all entities of a LP at simulation time T can only schedule new events with time stamp of at least T+L, then L is referred to as the lookahead for that LP. The Lower Bound Time Stamp (LBTS) of a LP is essentially the smallest timestamp of events that may be received in the future. The events before LBTS of a LP are referred as safe events which can be processed safely. If every LP process their safe events in parallel, then it is sufficient to ensure that the parallel simulation results will produce exactly the same results as the corresponding sequential simulation.

We have designed and developed a PDES simulation engine SUPE [13] which mainly focuses on multi-node environments. Both optimistic and conservative time management algorithms are supported by this simulation engine like BTW, BTB and TW. SUPE can also work on multi-core architectures, but it adopts the multi-process mode which cannot take full advantage of the performance of multi-core architectures. The architecture of SUPE is described in Fig.6.

The construction of SUPE simulation application includes four basic modules, simulation object module, simulation object manager module, event module and message module. Each LP contains these four modules. The objects in the simulation object module are instantiated as the entities in the simulation application. The simulation object manager module manages the simulation object according to the category, and is responsible for the creation and destruction of the simulation entity. The event module and the simulation object module are many to one relations, the event function in the simulation object is encapsulated into the simulation event for the global call through the macro definition. Message module is mainly responsible for the interaction between LPs. LPs across the nodes communicate through TCP/IP or MPI, and LPs in one node communicate through shared memory. The scheduling of events is realized by sending and receiving messages.

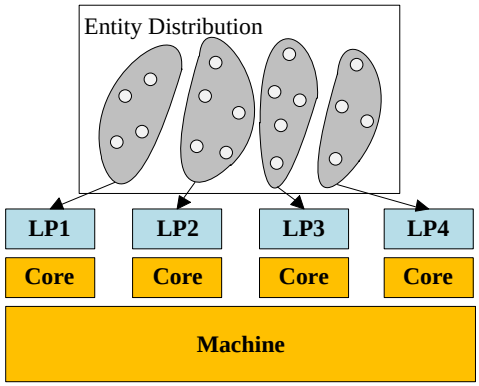

Figure 1: Architecture of simulation engine SUPE.

We can describe the execution flow of SUPE as follow steps. (1). Instantiate simulation objects and event encapsulation. The simulation object manager module instantiates each simulation object into multiple simulation entities. The event functions of the simulation object are encapsulated as event classes; (2). Entity Distribution. Distribute all instantiated simulation entities to the LPs. Distribution mode can be set to Block mode (entities of the same kind are assigned to one LP as far as possible), SCATTER (entities of the same kind are assigned to different LPs as far as possible) and custom mode (manual entity distribution); (3). Process events in parallel. Each LP processes their own safe events in time stamp order (4). Synchronization and GVT advance. Different time management algorithms have different synchronization conditions. For example, in the conservative time management algorithm, synchronization is conducted between LPs when all the events with the time stamps before LBTS are processed. During synchronization, every LP exchange their own local virtual time (LVT) and the current global virtual time (GVT) is calculated and broadcasted to all the LPs.

At present, almost all of the PDES simulation engines, including multi-node and multi-core platforms adopt this mode of execution. But some problems is difficult to resolve for this mode. First is the load balancing problem. As we all know, the computational complexity of simulation entities are different and difficult to predict, which cause the difficulty to divide the load into LPs evenly. Besides, computation hot spots may also migrate during running of simulation which cannot be handled by entity distribution. The second problem is communication and synchronization overhead. In order to maximize parallelization, we usually set the number of LPs to the maximum number of physical processes supported by the machine in simulation application. However, the increase of the number of LPs also brings greater synchronization and communication overhead due to the increased times of communications.

3.2 Fine Grain Parallelism

Aiming at the defects of traditional simulation engines, we design a fine grain parallel method to implement entity level parallelism in one LP. In time management algorithms, processing events within each LP in time stamp order can ensure the correctness of the simulation results. This is the theoretical basis of PDES. However, these algorithms ignore the events within the LP also can be processed in parallel. As we all know, each LP will process the safe events, of which the time stamps are before LBTS and LBTS is the minimum time stamp of events that the LP will receive in the future. This indicates that all the LPs will not schedule events before the time stamp of safe events of each LP during the execution cycle. And events of different entities of one LP can be processed in parallel, because the lookahead mechanism ensures that the time stamp of events each entity schedules is greater than that of all safe events of entities in this LP.

In order to make the LP internal safe events of different entities are processed in parallel, we need to manage safe events of different entities firstly. In traditional conservative time management algorithms, each LP stores all events of all entities in an event queue in the order of time stamp. Different from them, we propose a two-level event management structure which consists of entity management level and event management level for each LP. Entity management level manages the entities assigned to the LP. And in event management level, each entity manages its own safe events. With the two-level event management, we can easily access events of different entities and process them in parallel.

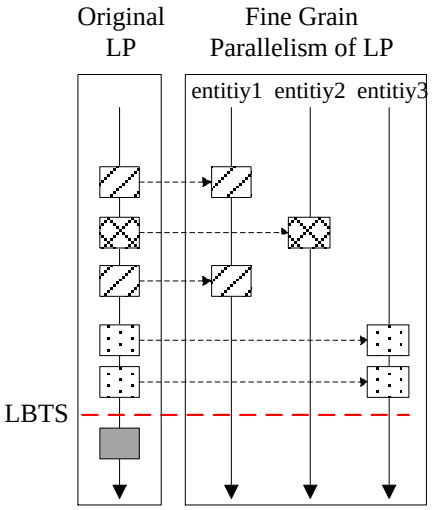

Figure 2: The process of fine grain parallelism of entities in one LP.

In each execution cycle, we dynamically schedule threads in the thread pool to dynamically process events according to the number of entities which currently have safe events to handle. The process is shown in Fig.2. When the number of entities in a LP is too many, the threads in the thread pool are all scheduled, and the idle threads take the initiative to get the events to be processed. In order to avoid memory access conflicts, we save the messages generated by each entity when processing events into the output message queue of this entity. The final step of event processing is sending out messages. In traditional PDES engines, the event messages must pass through the global message buffers. And even communication between entities of the same LP need to use global message buffers. Different from this, in our method, the messages generated by one entity will be inserted into the input message queue of the target entity directly if source entity and target entities are in the same LP. Because the memory are shared by all the entities of one LP and entities can interact directly between each other.

3.3 Multi-level GVT Synchronization

In the conservative event management algorithm, each LP maintains a local virtual time (LVT). After all LP processes the safe events, LPs will exchange the LVT value and set the current GVT of the system with the smallest LVT. Finally the GVT value will be broadcasted to all the LPs. Every will release unused memory according to GVT. Many efficient algorithms are proposed for the synchronization of GVT between LPs like Tree Barrier and Butterfly Barrier. However, events of entities within one LP are processed in parallel in our method, we need to synchronize LVT between entities of one LP.

By modify current parallel synchronization method, we design a multi-level GVT synchronization method which is suitable for the fine grain parallelism of LP. At first, we still apply the traditional parallel synchronization method between LPs. However, when it comes to one LP, we need to find out the minimum time stamp among all the time stamps of unprocessed events of all the entities and set LVT with the minimum time stamp after all the safe events are processed. The simplest method is traversing all entities to find LVT. But this method needs thread synchronization which may cause some threads to wait. And too many entities make the overhead of traversing cannot be ignored. Focusing on the problems, we propose a parallel LVT computing method using shared variables. In each LP, we define a shared variable which can be accessed by all the threads. Each thread will firstly record the minimum time stamp of events it processed. After processing all the safe events, each thread will check whether the LVT shared variable is locked. If the LVT shared variable is unlocked, the thread will lock this variable and update the LVT with the minimum time stamp it recorded when the value LVT is larger than the minimum time stamp. Finally, the thread will release the lock and other threads can access the LVT variable.

4 Evaluation

In order to evaluate the optimized method proposed in this paper, we apply the Pholds benchmark to test the performance from different aspects. By modifying the communication network structure and the amount of computation of entities in Pholds, we simulate some situations of the actual simulation applications to test the simulation engines. The hardware environment of all these experiments is a high performance multi-core machine with one way 2.93GHz Intel Xeon CPU X5670 and 24G RAM memory. The CPU contains 6 cores and each core contains 2 physical threads. The parallel discrete event simulation engine we optimized is SUPE, so we mainly compare the performance of our optimized engine to the original SUPE to evaluate our methods.

To test the performance of simulation engines when loads are unbalanced, we firstly design a Pholds application with the star structure. As shown in Fig.3, in the star structure, entities are divided into two parts, central part and the edge part. The number of entities in edge part is 10 times as large as that in central part. Entities in central part will interact with all entities but entities in edge part do not communicate with each other. What means the central part contains the greatest computational complexity.

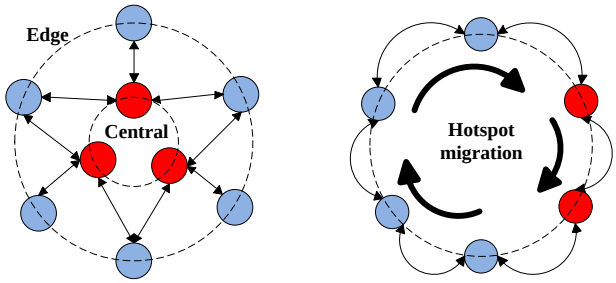

Figure 3: Two structures of the Pholds. (The left one is the star structure and the right one is ring structure)

Based on the star structure, we firstly discuss the impact of the number of processes and threads on the performance of the optimized simulation engine. The parameters we set for Pholds are as follows. We initialize 12 entities in central part and 120 entities in edge part. Every entity has 2 events at the beginning. We also set the lookahead to 0.1s and prove that time stamps of all new events are greater than current time stamp plus 0.1. Finally, the simulation end time is set to 20s. All the results are the average of multiple test results and are shown in Table 1. It is clear to find that the running time of Pholds is minimal when using all the 12 physical threads which is 2.5 times faster than using 4 physical threads. Besides, the results also show that the fewer LPs and the more threads bring higher performance when using the same number of physical threads. This is mainly because the increase of the number of LPs results in greater synchronization overhead.

Table 1: Runnnig time with different numbers of LPs and threads

Num of Physical Threads	Num of LPs	Size of Thread Pool	Running Time(s)
4	2	2	6.105
6	2	3	4.234
	3	2	5.255
8	2	4	3.079
	4	2	4.998
10	2	5	2.533
	5	2	4.805
12	2	6	**2.449**
	3	4	3.977
	4	3	3.84
	6	2	4.665

In this section, we also compare the performance of the optimized simulators with the original simulation engine when using different entity distribution modes including BLOCK mode and SCATTER mode. And to ensure that the two methods use the same physical computing resources, we set the number of LPs used in original simulation engine and the number of overall threads (the number of LPs multiplied by the number of threads in each LP) used in our optimized simulators to the same. We also discuss the influence of the number of process on the performance.

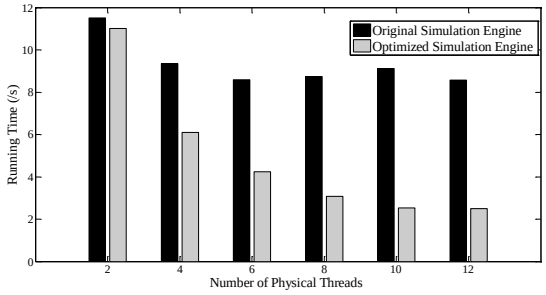

Figure 4: Comparison of running time of optimized and original simulators in BlOCK mode on the star structure Pholds

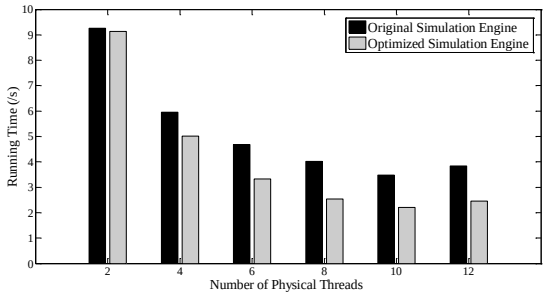

Figure 5: Comparison of running time of optimized and original simulators in SCATTER mode on the star structure Pholds

Fig.4 and Fig.5 shows that regardless of which entity distribution mode, the performance of the optimized engine is higher than the original simulation engine. Moreover, the greater the number of processes used, out optimized simulator has more advantages. Under the two entity distribution modes of BLOCK and SCATTER, our optimized simulation engine gains 3.5 and 1.5 times performance improvement, respectively. And it is obvious to find that acceleration effect is better in BLOCK distribution mode. Because the Pholds benchmark in this paper will assign the computing hotspots as far as possible into one LP in BLOCK mode comparing to SCATTER mode. In original simulation engine, when the computing hotspot is distributed to one LP, the parallelism performance is greatly reduced. However, with our optimization, the computing hotspot in one LP can also be processed in parallel with multi threads. So when the load distribution is unbalanced, our method is more advantageous.

Besides, we also find that when the load distribution is balanced, our optimized simulator also has a better performance comparing to the original engine. Because our approach can reduce the number of LP without reducing parallelism by using multi threads in each LP, which reduces the overhead of synchronization.

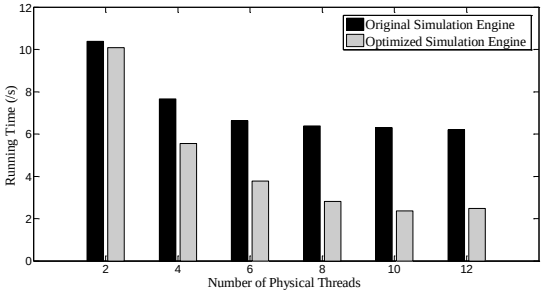

Figure 6: Comparison of running time of optimized and original simulators on the ring structure Pholds

In this section, we also designed a PHOLDS application in which the computing hotspots migrate dynamically to evaluate the performance of the optimized simulator. This Pholds application has a ring structure shown in Fig.3. Only the adjacent entities interact with each other and the Pholds will determine the number of events scheduling for each entity based on the current simulation time, thus allowing the computing hotspots to loop over the ring to simulate the behavior of load migration. We set the number of entities to 132 just like the number of overall entities in star structure. The other parameters are set the same as above experiments. Finally, we compare the performance of the original and the optimized simulation engines in this ring structure Pholds benchmark. The results shown in Fig.6 show that our approach can better adapt to the dynamic load migration.

5 CONCLUSIONS

In our research, we find that time management algorithms used in current parallel simulation engines ignore the parallelism of entities in one LP and the performance of current simulators relies heavily on entity distribution effects. Focusing on these drawbacks, an optimized approach which supports fine grain parallelism in one LP is proposed. At the same time, aiming at the new synchronization requirements, a multi-level GVT synchronization method is proposed. In experiment part, we designed Pholds with two structures (star and ring) to test the performance of the optimized simulators and compare it to the original simulation engine. Results show that our approach can improve the performance of the simulation engine greatly. Especially for the unbalanced load situations, the effect is much more obvious.

REFERENCES

[1] Miller, R. J. 2010. Optimistic parallel discrete event simulation on a beowulf cluster of multi-core machines .

[2] Wang, J., Jagtap, D., Abu-Ghazaleh, N., & Ponomarev, D. 2014. Parallel discrete event simulation for multi-core systems: analysis and optimization. J. IEEE Transactions on Parallel & Distributed Systems, 25(6), 1574--1584 .

[3] Yang, C., Li, B. H., Chai, X., & Chi, P. 2013. Ivy: a parallel simulator for variable structure systems under multi-core environments. J. International Journal of Service & Computing Oriented Manufacturing, 1(2), 103--123

[4] Yang, C., Chi, P., Song, X., Lin, T. Y., Li, B. H., & Chai, X. 2016. An efficient approach to collaborative simulation of variable structure systems on multi-core machines. J. Cluster Computing, 19(1), 29--46 .

[5] Das, S., Fujimoto, R., Panesar, K., Allison, D., & Hybinette, M. 1994. GTW:a time warp system for shared memory multiprocessors. Simulation Conference Proceedings (pp.1332-1339). IEEE.

[6] Lin, Z., Tropper, C., Ishlam Patoary, M. N., Mcdougal, R. A., Lytton, W. W., & Hines, M. L. 2015. NTW-MT: a Multi-threaded Simulator for Reaction Diffusion Simulations in NEURON. In: SIGSIM-PADS, pp.157--167.

[7] Glazer, D. W., & Tropper, C. 1993. On process migration and load balancing in time warp. In: IEEE Transactions on Parallel & Distributed Systems, 4(3), 318--327

[8] Peschlow, P., Honecker, T., & Martini, P. 2017 A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation. In: International Workshop on Principles of Advanced and Distributed Simulation, pp. 219--228. IEEE.

[9] Ahn, T. H., Sandu, A., Watson, L. T., Shaffer, C. A., Cao, Y., & Baumann, W. T. 2015. A framework to analyze the performance of load balancing schemes for ensembles of stochastic simulations. In: International Journal of Parallel Programming, 43(4), 59--630.

[10] Powley, C., Ferguson, C., & Korf, R. E. 1993. Depth-first heuristic search on a simd machine. J. Artificial Intelligence, 60(2), 199--242

[11] 15. Hillis, W. D. 1985 The connection machine. J. Scientific American, 267, 84--85

[12] Mermillod-Blondin, A., Stoian, R., Boyle, M. L., Rosenfeld, A., Burakov, I. M., & Audouard, E., et al. 2000. Parallel programming: techniques and applications using networked workstations and parallel computers. J. Landslides, 8(1-2), 391--401.

[13] Yao, Y. P., & Zhang, Y. X. 2008. Solution for analytic simulation based on parallel processing. In: Journal of System Simulation, 20(24), 6617--6621.