Procedia Computer Science

Volume 51, 2015, Pages 150–159

ICCS 2015 International Conference On Computational Science

Using high performance algorithms for the hybrid simulation of disease dynamics on CPU and GPU

Vasiliy N. Leonenko[1], Nikolai V. Pertsev[2], and Marc Artzrouni[3]

[1] ITMO University, Saint Petersburg, Russian Federation
vnleonenko@yandex.ru

[2] Sobolev Institute of Mathematics of Siberian Branch of the RAS, Novosibirsk, Russian Federation
homlab@ya.ru

[3] Universite de Pau et des Pays de l'Adour, Pau, France
marc.artzrouni@univ-pau.fr

Abstract

In the current work the authors present several approaches to the high performance simulation of human diseases propagation using hybrid two-component imitational models. The models under study were created by coupling compartmental and discrete-event submodels. The former is responsible for the simulation of the demographic processes in a population while the latter deals with a disease progression for a certain individual. The number and type of components used in a model may vary depending on the research aims and data availability. The introduced high performance approaches are based on batch random number generation, distribution of simulation runs and the calculations on graphical processor units. The emphasis was made on the possibility to use the approaches for various model types without considerable code refactoring for every particular model. The speedup gained was measured on simulation programs written in C++ and MATLAB for the models of HIV and tuberculosis spread and the models of tumor screening for the prevention of colorectal cancer. The benefits and drawbacks of the described approaches along with the future directions of their development are discussed.

Keywords: mathematical epidemiology, individual-based modeling, computer simulation, Monte Carlo methods, parallel computing, C++, MATLAB, OpenMP, MPI, GPGPU, CUDA

1 Introduction

In the contemporary mathematical epidemiology there exists a sufficient number of different modeling approaches, from populational modeling by means of analytically tractable ordinary differential equations to explicit compute-intensive agent–based modeling. Each of the approaches has its unique features, the question of choosing one for the particular scientific research is a matter of discussion [10]. In the last years a certain interest has arisen to the

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.214

employment of multicomponent models which combine several approaches in one composite model structure [2], [14].

In our works on disease propagation dynamics [5], [8] we came to use of two-component stochastic models incorporating a compartmental submodel and a discrete-event simulation. The employment of several submodels with different levels of detail allowed us to describe thoroughly the modeling processes of foremost importance and at the same time to stay reasonable in terms of computer power and amount of input data demanded for the sake of simulation. Still, the simulation time was becoming unsatisfactory when big populations were simulated or multiple launches with different input parameter sets were performed, that's why we have decided to turn to high performance algorithms. Various types of those are used to accelerate particular epidemic models and modeling frameworks [3], [4], also they are often included into software packages for epidemic modeling [11], [15].

The important issue is that the researcher's efforts spent on converting the serial modeling algorithm to parallel one is not always compensated by the resulting overall time economy, especially when (a) the model is not planned to be reusable (i.e. it is built to address particular research issues and will be discarded after several experiments) and (b) the serial algorithm speed seems more or less satisfactory to the researcher. Both (a) and (b) hold true in our case, so we've decided that performing tedious algorithm optimization for every particular model is not worthwhile. However, we have found out that a number of general high performance techniques could bring a speed boost for a big class of our models (including the existing ones and those which could be created later) and hence compensate the time spent on the algorithm modification. In this paper we introduce three possible approaches for the simulation speedup with different tradeoff between the algorithm universality, performance and easiness of implementation.

2 Two-component Individual-based Models

Population structure. The population under study is regarded as an aggregate of groups $C = \{A_1, A_2, \ldots, A_n\}$ reflecting spacial and social heterogeneity along with difference in disease states of individuals. We distinguish a subset of our aggregate C, $F \subseteq C$, which contains groups of individuals affected by the disease. While all the individuals within a state $A_i \in C \backslash F$ are considered indistinguishable, each individual x in a fixed state $A_i \in F$ has a certain value of the continuous non–negative parameter (a scalar or a vector) which changes via time according to some deterministic or stochastic law and corresponds to the extent of disease impact on the individual. The interpretation of this parameter depends on the regarded disease and the particular model: it could be tumor size, virus density in blood or a unit–less value reflecting the disease burden. The quantities of individuals in each state change due to processes of birth, death, migration and transitions from one state to another. All these processes are considered stochastic and described by random variables.

Submodel interaction. The compartmental submodel is in charge of processes which are connected with individuals within an aggregate $C \backslash F$, as well as with the flows to all of the groups (in general case it includes the processes of birth, migration, death from natural causes and infection of susceptible individuals). The submodel is set by a discrete or continuous-time Markov chain or a system of stochastic difference equations on group quantities for the moments $t = 1, 2, \ldots, T$. The discrete-event submodel is used to simulate the transition between the groups of F independently for every single individual $x \in A_i$, $A_i \in F$. The starting condition of discrete-event simulation is the appearance of an individual x in the group $A_i \in F$ due to birth,

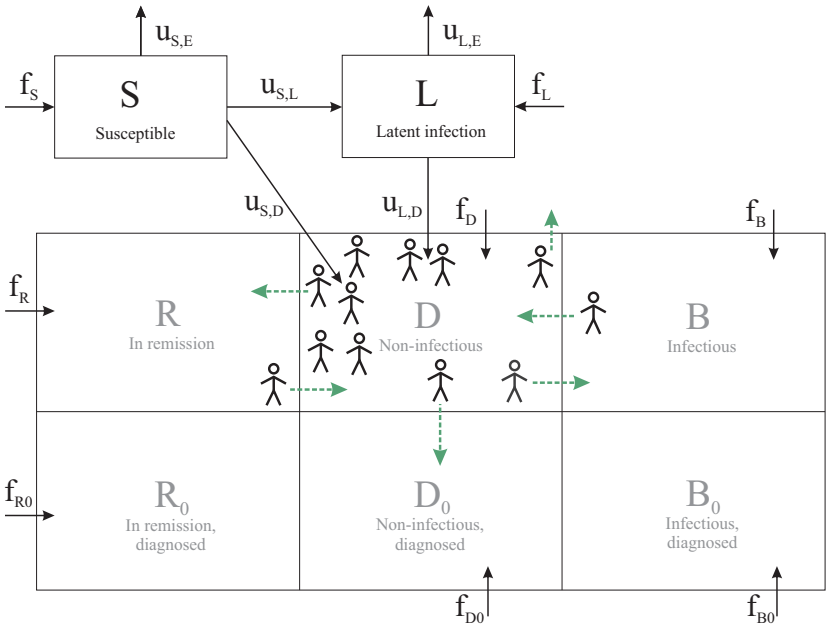

Figure 1: A two-component model of tuberculosis propagation from [8]. The compartmental submodel handles two groups: susceptible individuals (S) and individuals with latent infection (L). The aggregate F consists of six groups reflecting different disease stages (non-infectious, infectious, in remission) and treatment status (diagnosed or not). The individuals acquire the infection after contacting infectious individuals from the groups B and B_0.

migration or transition from the state $A_j \in C \backslash F$. The ending condition is reached when the individual x either moves to a group $A_j \in C \backslash F$ or entirely leaves the population. An example of the two-component model is shown on figure 1. If in the particular research case we don't need to consider individual parameters connected with disease advance, then $F = \varnothing$ and the model becomes purely compartmental. In some other cases, on the contrary, we could assume that $F = C$ and therefore work with purely discrete-event model — it's convenient, for instance, when we investigate the disease onset and advance within a fixed size group of individuals of the same age, without birth and migration flows.

Output. The conventional output data includes the quantities of individuals in each group at the fixed time moments $t = 1, 2, \ldots, T$. Additional output could be studied according to the research task: it could be number of deaths in population during modeling time, potential years of life lost, economical cost of medical interventions, etc. If the model is purely compartmental, in some cases the values of output variables can be assessed analytically. For instance, discrete-time compartmental models can be analyzed with the help of the systems on mathematical expectations of group quantities which are used for searching the equilibrium states [9] or building of deterministic approximations of the group quantities dynamics [8]. For continuous-time compartmental models deterministic approximations could be used to study the stationary probabilities distribution of a corresponding continuous-time Markov chain [1]. The mentioned techniques are not applicable to individual-based models, so in general case

one needs to estimate the expectations and variances of the observed random variables with the help of Monte Carlo methods which requires launching the modeling program several times with different initial seeds for the random number generator.

Serial algorithm structure. Suppose one numerical experiment consists of N simulation runs and the modeling time interval is T. The general scheme of the algorithm could be presented as follows.

A. Preparing the array R of initial seeds for the random number generator

B. for $i = 1, 2, \ldots, N$

  B.1. Setting an initial seed to $R[i]$

  B.2. for $t = 1, 2, \ldots, T$

    B.2.1. Finding the migration flows, transition flows between the groups $A_j \in C \backslash F$ and from $A_j \in C \backslash F$ to $A_k \in F$ during the time period $(t-1; t]$, using the compartmental submodel

    B.2.2. For every individual x newly appeared in A_k, $A_k \in F$:

      B.2.2.1. Simulating state transitions and changes of value $\Delta_x(t)$ using the discrete-event submodel

  B.3. Storing output variable values

C. Assessing the mathematical expectations and variances of output variables by the sample of N values calculated during the simulation runs

In case if we perform uncertainty/sensitivity analysis of our model on M input parameter sets, the algorithm will be implemented consequently M times for every set of initial values.

3 Parallel And Distributed Algorithm Approaches

In this section we describe the ideas of parallel algorithms and present their relative performance for the particular models compared to the serial algorithm. The modeling programs used in the experiments were implemented in MATLAB (a simulation program for the continuous-time model of HIV spread between intravenous drug users [1]) and C++ (all the other programs mentioned in this work). In each of the experiments the simulations were performed several times and the best simulation time was considered for the speedup assessment. As the baseline we used the serial modeling programs with Mersenne Twister random number generator.

Name	CPU type	Number of cores	RAM, Gb	GPU type
desktop	Core i7-870 2.93 GHz	4	4	GeForce GTS 450 Ti
laptop	Core i5-4210U 1.7 GHz	2	8	GeForce GTS 730M

Table 1: Computer configurations used for the experiments

3.1 Batch Random Number Generation on CPU and GPU

Method description. Due to the nature of the algorithms based on Monte Carlo methods they require a lot of random numbers to operate with, so the increased speed of delivering random numbers could result in dramatic acceleration of simulation process. As far as many contemporary random number generators have fast methods for generating random numbers in chunks, one of the possible means to achieve a speedup is to employ batch random number generation by means of CPU or GPU instead of generating single random numbers "on demand". The drawback is the increased memory demand of the modified algorithm, as the arrays of random numbers are to be stored in the memory.

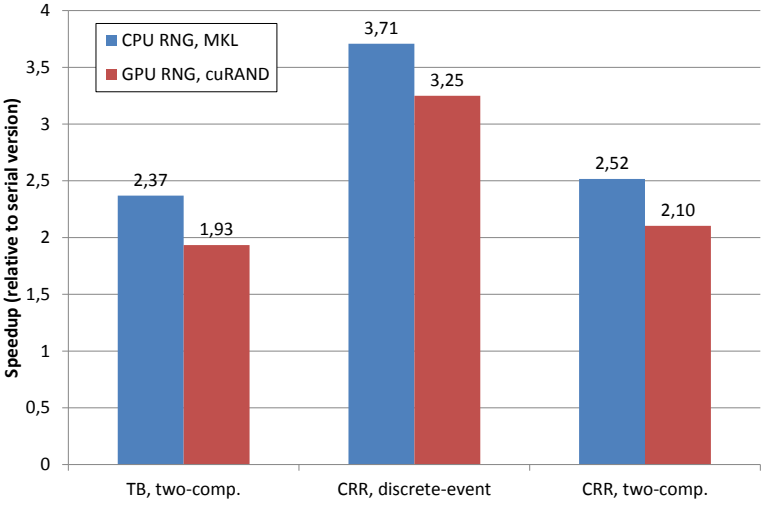

Figure 2: A speedup for different model types received by employing batch random number generation on CPU and GPU

Numerical results. In Fig. 2 the speedup results are shown for the simulation programs of three models: the two-component individual-based model of tuberculosis propagation [8], the discrete-event model of tumor screening for the prevention of CRR [7] and its two-component version [5]. For the batch random generation on CPU we used Mersenne Twister generator from Intel Math Kernel Library, the class for batch random number generation on GPU was created using NVIDIA CUDA technology and XORWOW generator from cuRAND library (CUDA SDK). To store random numbers in computer memory a 2D array with the size 3×10^8 was employed in each case. Every time when all the random numbers from the current 1D chunk of size 10^8 were spent, the chunk was asynchronously refreshed by generating new random numbers. The simulations were performed on both hardware configurations listed in Table 1, however, as far as the resulting speedup didn't depend much on the computer type, to avoid redundancy we present the graphs only for the simulations performed on the laptop.

As we see from Fig. 2, we receive significant performance increase with batch random number generation both on CPU and GPU. On the exploited hardware configurations Intel MKL outperforms cuRAND. Random number generation on GPU could dramatically outperform that on CPU in case if the hardware configuration with a powerful graphical processor unit is

used for the simulation [6], but the unavoidable operation of copying the random numbers from GPU memory to RAM could diminish the difference in hardware performance, which apparently happened in our case. So whether to choose batch random number generation on CPU or GPU depends much on the configuration of the system. We could dare to presume that for the majority of personal computers with a balanced hardware configuration using CPU for random number generation should be slightly more effective.

It also came to fact that the same technique applied to MATLAB HIV modeling program doesn't affect positively on the simulation speed, on the contrary, the calculation time increased both for CPU and GPU batch random generation compared to the initial algorithm. The program profiling showed that the random number generation is not a bottleneck for the performance of our particular MATLAB simulation. The explanation could be in difference of operation costs in MATLAB and C++ or in the fact that our serial MATLAB code has flaws and should be refactored before we try to receive benefits from the batch number generation.

3.2 Distribution of Simulation Runs

Method description. As far as the numerical experiment usually consists of performing several simulation runs (the typical number is $N = 100$) which are executed independently from each other and don't require data transfer from one to another during the calculations, the calculation speed could be increased by the distribution of simulation runs among the available computational nodes. The resulting algorithm doesn't depend on the structure of a particular model and on the number of its submodels.

To implement this approach in our C++ simulations we modified the serial algorithm using MPI and OpenMP technologies. The former is suitable for the computers with shared and distributed memory, whereas the latter could work only in shared memory. In MATLAB program we used directives for shared memory algorithms from MATLAB Parallel Computing Toolbox.

To distribute simulation runs between the computational cores with the help of OpenMP or MATLAB Parallel Computing Toolbox the only thing to be done is to parallelize the cycle in line B of the serial algorithm (see section 2); `#omp parallel for` clause was used for that purpose in C++ OpenMP programs and a similar `parfor` directive in MATLAB modeling program. In MPI programs the delivering of input parameter values from the main thread to the others should be provided, for that purpose a directive `MPI_Send` was put between the lines A and B. After the simulation, before executing the line C, the main thread gathers the output data with the help of directive `MPI_Recv`.

Numerical results. We measured the speedup for the simulation programs for compartmental and two-component individual-based models of tuberculosis propagation [8], the compartmental model of HIV propagation considering the social disadaptation of different population groups [9] and the MATLAB HIV model. The experiment was held on the "desktop" configuration (see Table 1), as it has twice as much physical cores as the laptop. The results are shown in figure 3. It's worth noting that the MPI algorithm used for the two-component TB model shows lower speedup on 2 threads than its shared memory counterparts (C++ OpenMP and MATLAB parfor) — that could be caused by the necessity for an MPI algorithm to spend additional time on creating the copies of input variables and gathering the output values. At the same time, the scalability of MATLAB and OpenMP programs doesn't seem to be satisfactory, which makes it possible for the MPI program to win in speedup on 8 threads. Possibly as the number of threads increases, they start to hinder each other when working with the shared data, while the MPI threads work with independent copies of variables and thus don't have

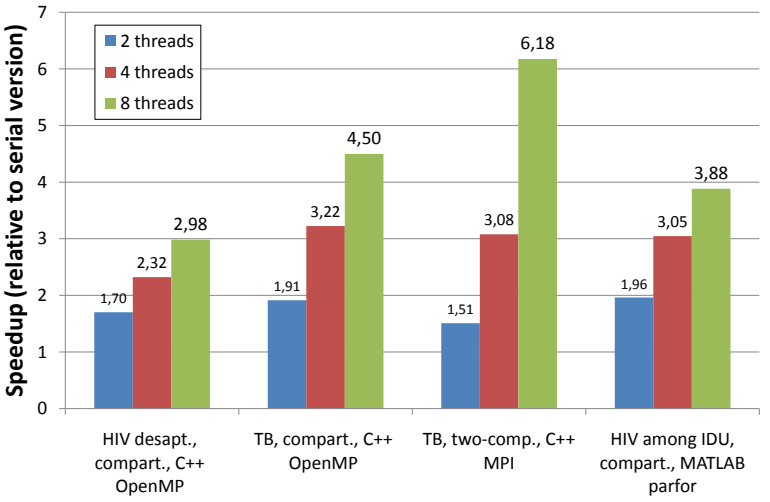

Figure 3: A speedup of simulation program versions for different models gained using OpenMP and MPI technologies

this problem. Further investigation is planned to solve the issue and to make OpenMP and MATLAB programs more scalable.

3.3 Simulation on GPU

As we saw in the section 3.1, a modern GPU has a big potential for general-purpose calculation, but the simulation performance could be reduced due to necessity of sending data from GPU to CPU. One of the ways to avoid unnecessary transactions between the CPU and the GPU is to move the majority of the calculations to GPU. Due to their hardware structure GPUs are especially good at performing massive-parallel operations when a small set of instructions ("a kernel", according to NVIDIA terminology) is applied to big arrays of data [12]. That gives us a lot of "lightweight" identical threads being launched simultaneously. The technique of simulation runs distribution which we used in the previous section to perform calculations on multi-core CPUs is not applicable here in general case, because a typical number of simulation runs is only about 10^2, and, which seems even more important, one simulation run is usually too "heavy" for a single GPU computational core in a sense of memory and processor power consumption. The ways to overcome this obstacle and bring GPU into play for some classes of models are discussed below.

Discrete-event models of non-infectious diseases. Assume we regard an onset and advance of some non-infectious disease in a group of m individuals of the same age. In this case we can simulate the modeling processes (death, aging and disease occurrence and advance) for any individual separately from others, hence a GPU kernel function will be in charge of modeling the whole life of a fixed individual from the beginning of the simulation to its end. The cycles at line B and B.2.2. of the serial algorithm are merged and replaced by one cycle with $N \cdot m$ iterations, which gives us the number of parallel threads sufficient to benefit from massive-parallel calculations on GPUs.

Two-component models of non-infectious diseases. Let's now assume that we start to consider the processes of birth and migration, i.e we regard a full-fledged population which requires a separate populational submodel. As far as the disease under study is non-infectious, still all the processes happening with an individual are linear and their progress is not affected by the behavior of the other individuals. So the modeling procedure in this case could have the following algorithm:

• We serially launch N simulation runs on CPU to perform compartmental submodel which is in charge of the processes of birth and immigration for the given population.

• At the end of each simulation run we calculate the overall number of individuals $m = m(i)$, $i \in \overline{1, N}$, appeared in the population during that run.

• We launch in parallel the modeling processes for disease onset and progression for all $\sum_{i=1}^{n} m_i$ individuals simultaneously in a same fashion as it was done for the discrete-event model from the previous paragraph, the only difference is that now every individual has two additional parameters: a year of his arrival in the population and the index of simulation run.

• After getting the output results, such as the number of individuals died from the disease, we distribute the statistics by years and simulation runs to assess the mathematical expectations of the output variables.

Sensitivity analysis of compartmental models of infectious diseases. If the disease we regard is infectious, the model should incorporate the infection process, which is non-linear, so we cannot handle each individual in a separate thread from the beginning to end. That fact makes it impossible to use the parallel computing approach described in the previous paragraph. Creating of a general GPU simulation algorithm for our two-component models still remains a challenge (and the question whether such an algorithm exists at all is arguable).

So far we've managed to employ GPU for the sake of sensitivity analysis of the compartmental models. In this situation we need to perform N simulation runs for the each of M input parameter sets which gives us $M \cdot N$ independent threads. The function which is performed on the data is one simulation run, just like in section 2 — in case of compartmental models, unlike for two-component ones, it's often "lightweight" enough to be performed on GPU computational core. If this is not the case, or the environment for some reasons doesn't allow to present a whole simulation run as a kernel [1], we can split a simulation run into consequent time steps and regard each one (as the model is purely compartmental, it's the line B.2.1 of the serial algorithm) as a kernel. As a result the massive-parallel operations will be performed consequently in a loop with index $t = 1, 2, \ldots, T$.

Numerical results. We've applied three techniques described in the previous paragraphs to the discrete-event model of colorectal cancer screening, two-component model of colorectal cancer screening and HIV MATLAB model correspondingly. While in the first case the initial algorithm was only slightly redesigned, in the second and third cases we had to rewrite the algorithms sufficiently to make them suitable for massive-parallel calculations, which by itself gave a 1.79–2.22 speedup for the two-component colorectal cancer model and 1.26–1.51 speedup for the HIV model depending on the hardware configuration. These modified algorithms were taken as a baseline. To move the calculations onto GPU we employed NVIDIA CUDA SDK in C++ models, in MATLAB the same operation was made using `gpuArray` and `arrayfun` functions (see [13]). The resulting speedup is shown on figure 4. As we see, the speedup values for the purely discrete-event model and the two-component model are close to each other, thus we can conclude that serial CPU calculations performed for the sake of compartmental

[1] We've faced this limitation in MATLAB while employing GPU calculations with **arrayfun** function

submodel don't affect much the algorithm scalability. The speedup for the MATLAB program is not as inspiring, as in the case of colorectal cancer models (although it's still sufficient), apparently because of the necessity to perform consequent massive-parallel operations with transferring the output to computer memory and back at every time step. We could increase the simulation speed by embedding a kernel written in CUDA C into MATLAB program [13] instead of using `arrayfun` function, but it requires additional programming we aimed to avoid from the beginning (see "Introduction").

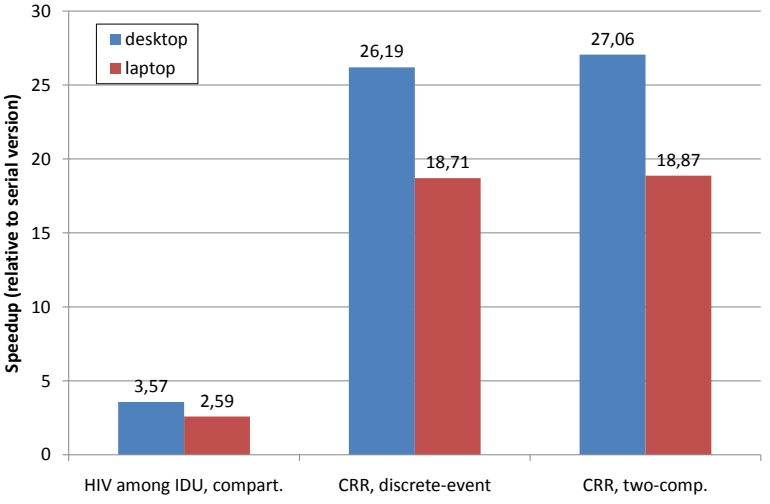

Figure 4: A speedup of GPU simulation program versions

4 Conclusions and Perspectives

In this paper we have presented three parallel simulation approaches applicable for two-component disease propagation models, and discussed their benefits and drawbacks. As one can see, if the model under study doesn't exhibit high level of detail, it seems reasonable to apply general high performance approaches without thorough program code refinement and optimization (like those based on simulation runs distribution or batch random number generation). Practically at no costs, merely by adding a few lines of code into the existing serial programs, we could sufficiently reduce the simulation time which is always convenient even in case if the serial simulation is not dramatically slow. For more compute-intensive simulations the approaches involving massive-parallel computations on GPUs could be brought into play, but that requires more thorough code refactoring. It's up to the researcher to decide which approach to choose in his modeling case.

At the moment of writing this article we are satisfied with the acceleration received with the help of approaches described in the paper. If our future research tasks require more sophisticated and time-consuming simulations, we find reasonable to drift towards coupling several high performance approaches within one simulation, for instance, using OpenMP+GPU on personal computers and MPI+OpenMP+GPU on supercomputer clusters. The prospected work in this direction requires solving the technical issues of approaches interconnection (such as deciding

what algorithm parts are to be performed on CPUs and what are for GPUs) and investigating new ways of GPU employment to make it possible to cover greater number of modeling cases. Still, we don't want to forget about the algorithm universality, so we hope to make these coupled approaches suitable for a variety of models without considerable tuning in every particular case.

5 Acknowledgements

The authors are thankful to the two unknown referees whose thoughtful remarks made it possible to find drawbacks in experiment procedures and to improve significantly the quality of the paper.

This paper is supported by Russian Scientific Foundation, grant #14-21-00137 "Supercomputer simulation of critical phenomena in complex social systems". The research is done in Advanced Computing Lab (ITMO University), which is opened in frame of 220 Decree of Russian Government, agreement #11.G34.31.0019.

References

[1] M. Artzrouni, T. Mara, and V. Leonenko. A syringe-sharing model for the spread of HIV: application to Omsk, Western Siberia with a sensitivity analysis. Submitted for publication, 2015.

[2] B.J. Binder, J.V. Ross, and M.J. Simpson. A hybrid model for studying spatial aspects of infectious diseases. *The ANZIAM Journal*, 54(1-2):37–49, 2012.

[3] K.R. Bisset et al. INDEMICS: An interactive high-performance computing framework for data-intensive epidemic modeling. *ACM T. Model. Comput. S.*, 24(1):4, 2014.

[4] D. L. Chao et al. FluTE, a publicly available stochastic influenza epidemic simulation model. *PLoS Comput. Biol.*, 6(1):e1000656, 2010.

[5] V. N. Leonenko and N. V. Pertsev. Efficiency analysis of the programs of exposure of individuals predisposed to colorectal cancer based on imitational modeling. *Upravlenie Bol'shimi Sistemami*, 35:207–236, 2011. In Russian.

[6] NVIDIA. CURAND // NVIDIA CUDA Zone. [online]. `https://developer.nvidia.com/cuRAND`.

[7] N. V. Pertsev, E. Yu. Khomutova, and V. N. Leonenko. Application of mathematical modeling for estimation efficiency of CTC in detecting individuals predisposed to colorectal cancer. *Meditsinskaya Vizualizatsiya*, 2:104–108, 2011. In Russian.

[8] N. V. Pertsev and V. N. Leonenko. Stochastic individual-based model of spread of tuberculosis. *Russ. J. Numer. Anal. M.*, 24(4):341–360, 2009.

[9] N. V. Pertsev and V. N. Leonenko. Discrete stochastic model of HIV infection spread within a heterogeneous population. *Russ. J. Numer. Anal. M.*, 27(5):459–478, 2012.

[10] H. Rahmandad and J. Sterman. Heterogeneity and network structure in the dynamics of diffusion: Comparing agent-based and differential equation models. *Manag. Sci.*, 54(5):998–1014, 2008.

[11] P. Richmond et al. High performance cellular level agent-based simulation with FLAME for the GPU. *Brief. Bioinform.*, 11(3):334–347, 2010.

[12] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[13] J.W. Suh and Y. Kim. *Accelerating MATLAB with GPU Computing: A Primer with Examples*. Newnes, 2013.

[14] M. Tizzoni et al. Real-time numerical forecast of global epidemic spreading: case study of 2009 A/H1N1pdm. *BMC medicine*, 10(1):165, 2012.

[15] E.A. Wenger and P.A. Eckhoff. A mathematical model of the impact of present and future malaria vaccines. *Malar. J.*, 12(126):10–1186, 2013.