Biophysical Journal Volume 108 February 2015 457–461

Computational Tools

ReaDDyMM: Fast Interacting Particle Reaction-Diffusion Simulations Using Graphical Processing Units

Johann Biedermann,[1] Alexander Ullrich,[1] Johannes Schöneberg,[1] and Frank Noé[1,*]
[1]Department of Mathematics, Computer Science and Bioinformatics, Free University of Berlin, Berlin, Germany

ABSTRACT ReaDDy is a modular particle simulation package combining off-lattice reaction kinetics with arbitrary particle interaction forces. Here we present a graphical processing unit implementation of ReaDDy that employs the fast multiplatform molecular dynamics package OpenMM. A speedup of up to two orders of magnitude is demonstrated, giving us access to timescales of multiple seconds on single graphical processing units. This opens up the possibility of simulating cellular signal transduction events while resolving all protein copies.

INTRODUCTION

Modern and quantitative experimental techniques have provided overwhelming evidence that cells are not well-mixed containers: specific spatial arrangements of molecules with well-defined particle numbers, the diffusion-mediated transport between them, which is often affected by crowding and space exclusions, and stoichiometric association/dissociation reactions play a key role in cellular signal transduction. No experiment is able to simultaneously probe localizations and time-sequences of all relevant molecular processes in vivo. Thus, computer simulations are unique in their ability to integrate available experimental data in a physically realistic model that can be run to study the complete cellular system.

Classical reaction kinetics simulation techniques lack the level of detail necessary to simulate biological systems at physically realistic conditions: well-mixed approaches do not resolve space, and concentration-based approaches do not keep track of molecule copy numbers. Among the most realistic reaction kinetics models are particle-based reaction-diffusion techniques that resolve every reacting molecule in space and time (1–3). Most available particle-based reaction-diffusion techniques lack interaction potentials (or forces) between particles that are required to realistically model molecular aggregates, clusters, or fibers/chains. Molecular dynamics (MD) simulations, on the other hand, permit modeling of particles with interaction potentials, but MD implementations are generally not built to facilitate reactions between particles and do not tolerate significant changes in the particle numbers over time.

The software package ReaDDy (4) is believed to be the first generic implementation of a new class of reaction kinetics models combining particle-based reaction-diffusion with interaction potentials, which we shall call interacting-particle-reaction-diffusion dynamics. In ReaDDy, one typically simulates signaling cascades by resolving each protein copy with a single or a few tethered particles. With most cellular proteins having micromolar concentrations, and most biochemical pathways requiring a few protein types, this leads to a few 1000 particles in a typical simulation setup. ReaDDy is a MD package in the sense that arbitrary interaction potentials can be defined between particles and multiple dynamical models can be employed, although time-discretized Brownian dynamics and Metropolis Monte Carlo are the most common models to update particle positions. ReaDDy also permits unimolecular and bimolecular reactions to be defined between any of the simulated particles (e.g., $A + B \rightleftharpoons C$). More complex reactions (e.g., three or more educts) must be split into bimolecular steps.

The reference implementation of ReaDDy, available under the BSD license at https://github.com/readdy, is written in JAVA, a language that combines rapid development with an execution speed similar to the language C. However, JAVA is less suited for highly parallel computing, and this single-threaded implementation practically limits simulation timescales of ~100 ms for ~1000 particles. These timescales facilitate the study of very fast processes, such as the activation phase in phototransduction requiring ~200 ms of simulation time with 1000 slowly-diffusing particles (5). To access a wider range of biological phenomena and larger simulation systems, a significant speedup is needed.

ReaDDy was designed with modular software architecture. In particular, its core package, which runs the most expensive computations, can be replaced by implementations in other languages that are optimized for specific computing platforms. Implementations based on the languages C or C++ can be easily executed through the

Submitted November 4, 2014, and accepted for publication December 4, 2014.

*Correspondence: frank.noe@fu-berlin.de

Editor: Daniel Beard.

0006-3495/15/02/0457/5 $2.00
http://dx.doi.org/10.1016/j.bpj.2014.12.025

JAVA native interface (JNI). To our knowledge, we present a new implementation of the ReaDDy core where the particle dynamics are propagated on graphical processing units (GPUs). This implementation uses the fast, open-source, and flexible MD package OpenMM (6,7) in order to propagate particle positions (here using Brownian dynamics although other integrators could be easily used), while computing the reactions on the central processing unit (CPU). It is demonstrated that this implementation can yield up to a factor-of-100 speedup when the timescales of reactions are significantly smaller than the particle integration timesteps. Such a scenario is common, with integration timesteps typically on the nanoseconds timescales, while biochemical reactions rarely have rates higher than $1/\mu$s. The two-orders-of-magnitude speedup will permit a much wider range of biological processes to be simulated at single-particle resolution with custom hardware, including the photoactivation in a complete rod cell, or presynaptic exo- and endocytosis in neurotransmission. Further applications of ReaDDy are discussed in Schöneberg et al. (8).

MATERIALS AND METHODS

ReaDDy and ReaDDyMM

Let

$$X(t_1,t_2) = \left(x_1(t_1,t_2),\ldots,x_{N_{t_2}}(t_1,t_2)\right)$$

be an ordered list describing the system configuration. We call t_1 the dynamics time and t_2 the reaction time—these two clocks are distinguished in order to represent the fact that the dynamical equations and the reaction kinetics equations are integrated separately. N_{t_2} is the total number of particles at reaction time t_2 (the length of X can vary over time). The value $x_i(t_1,t_2)$ is a vector containing all properties needed to describe the instantaneous state of particle i, at least three floating-point numbers for its position, and an integer number for its type, as

$$x_i(t_1,t_2) = \left[q_{i,1}(t_1,t_2),\ q_{i,2}(t_1,t_2),\ q_{i,3}(t_1,t_2),\ a_i(t_2)\right].$$

The vector could contain additional information, such as particle velocities when nonoverdamped dynamics are used. ReaDDy usually propagates the configuration by iterating two steps:

Step 1. Dynamics: $X(t,t) \rightarrow X(t+\tau,t)$, and
Step 2. Reactions: $X(t+\tau,t) \rightarrow X(t+\tau,t+\tau)$.

Particle reactions in ReaDDy can be very complex. For example, it is possible for a protein to consist of several tethered particles, and when binding a ligand particle this could be modeled as building a new particle into the particle network representing the protein-ligand complex. Thus, particle reactions may involve changes not only in the particles' numbers and positions, but also to the system topology and energy function. For this reason, ReaDDyMM was designed to handle all reactions within the ReaDDy CPU implementation (Fig. 1 *B*), while the computationally most intensive part of integrating the particle dynamics is delegated to the OpenMM GPU implementation (Fig. 1 *D*).

The setup and parameterization of the simulation occurs completely via the usual ReaDDy input files (Fig. 1 *A*). In particular, the user can specify various particle interactions, reaction rates, initial coordinates, particles diffusion constants, and integration timesteps for the reactions and dynamic calculations. Parsing this input and setup of the OpenMM simulation is entirely automated and invisible to the user. At initialization time, all particle positions and parameters such as radii and diffusion constants are passed from JAVA-ReaDDy (Fig. 1 *B*) to a C++ wrapper (Fig. 1 *C*), which builds the relevant OpenMM simulation objects. OpenMM was accessed using its C++ interface from JAVA using the JNI. After building the OpenMM system (Fig. 1 *D*), the interface calls a number, n, of integration steps. The value n is the coarse-graining factor for the reaction kinetics timestep, a simulation parameter that trades between speedup and accuracy of the simulation. Briefly, the dynamics/reactions integration scheme becomes

$$\text{GPU}: \ X(t,t) \rightarrow X(t+\tau,t) \rightarrow \cdots \rightarrow X(t+n\tau,t).$$

$$\text{CPU}: \ X(t+n\tau,t) \rightarrow X(t+n\tau,t+n\tau).$$

At time t, particle positions are copied from the CPU to the GPU in order to start the particle dynamics integration sweep. After n dynamics timesteps and a time $n\tau$ later, they will be copied back in order to evaluate and execute particle reactions on the CPU. Each such copy event involves data transfer through the PCIe bus and the JNI interface (Fig. 1, *B–D*). When n is small, these interfaces become a bottleneck in execution speed, and in the worst case could lead to a situation where the GPU implementation is slower than the CPU implementation. When n is large, a large speedup of the GPU implementation over the CPU implementation is possible. However, n can only be increased to a point before the reaction kinetics becomes inaccurate. The value $n\tau$ needs to be significantly smaller than the timescale of the fastest reactions in the system to avoid missing reaction events by too-large-reaction timesteps.

Because OpenMM does not support on-the-fly changes of particle numbers, we have introduced an initially noninteracting dummy particle buffer to generate or consume particles consumed or generated by reactions. Such a buffer is created for every particle type with a given diffusion constant.

RESULTS

We demonstrate the validity and efficiency of ReaDDyMM on a test system established in Schöneberg and Noé (4). We use a cubic box with 100-nm edge length. Particles interact with each other and with the box walls with a soft-core repulsion potential. We have simulated different numbers of particles: 1135, 3017, and 4733, to obtain systems with 10, 30, and 50% occupied volume density, respectively. Three different particle types (*A*, *B*, and *C*) are simulated with the reversible reaction $A + B \leftrightharpoons C$. (See Schöneberg and Noé (4) and Table 1 for the simulation parameters.) For the 50% density system we here used several sets of (forward $r_{A+B\rightarrow C}$ and backward $k_{C\rightarrow A+B}$) microscopic reaction rates: slow ($10^5\ \text{s}^{-1}$, $5\times10^3\ \text{s}^{-1}$), intermediate ($10^6\ \text{s}^{-1}$, $5\times10^4\ \text{s}^{-1}$), and fast ($10^7\ \text{s}^{-1}$, $5\times10^5\ \text{s}^{-1}$). The 10 and 30% systems were only simulated with the intermediate rates. The forward rates refer to reaction radii of 1.5 nm for *A*- and 3.0 nm for *B*-particles.

All systems were equilibrated using ReaDDy's Metropolis Monte Carlo implementation (4) before starting the production runs. The timestep for calculating the dynamics was kept constant at 0.1 ns. The reaction timestep was varied in order to explore the tradeoff between speedup and reaction kinetics accuracy. For each system, we computed

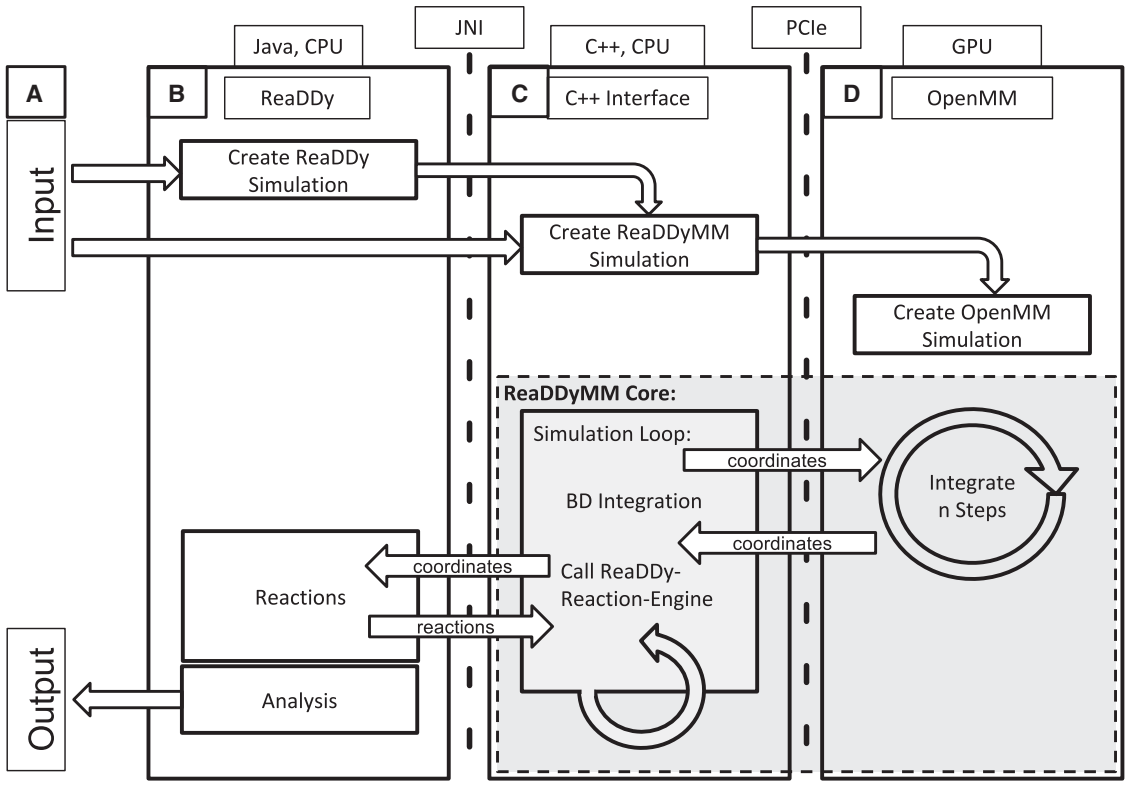

FIGURE 1 Scheme of the ReaDDyMM software architecture. (*A*) User layer: the user specifies particle interactions, reactions, and initial coordinates by input files; runs a ReaDDy simulation; and harvests trajectory data or statistical analyses. (*B*) Reaction kinetics layer: the reaction kinetics is handled by the JAVA-based CPU-implementation of ReaDDy. (*C* and *D*) Core layer: particle dynamics and simple reaction event handling is done by the modular ReaDDy core. This implementation uses OpenMM for the GPU implementation of particle dynamics (*D*), and connects to the reaction kinetics layer through a C++ interface (*C*). For a good speedup, it is important that a significant number n (typically 100–1000) of dynamics times steps are conducted per reaction timestep.

the kinetics by averaging over 10 independent runs of 0.01-ms length each.

Fig. 2 shows simulation results for a reaction timestep of $n\tau = 0.1$ μs that is obtained for $n = 1000$ dynamics integration steps per reaction step. For this setting, the results in Fig. 2 demonstrate that ReaDDyMM accurately reproduces the dynamics and reaction kinetics using the ReaDDy implementation with $\tau = 0.1$ ns as a reference.

TABLE 1 Depiction of the speedup from ReaDDy to ReaDDyMM

N	ρ (%)	$n\tau$ (μs)	t_0 (min)	t_{MM} (min)	Speedup
1135	10	0.1	100	3.5	28.4
3017	30	0.1	651	9.9	65.6
4733	50	0.01	1573	38.2	41.2
4733	50	0.1	1573	19.7	80.0
4733	50	1	1573	19.3	81.4
4733	50	n/a	1517	13.2	114.9

The dynamics timestep was chosen to be $\tau = 0.1$ ns. N is the number of particles simulated, ρ the particle density. The reaction timesteps $n\tau$ are given, and were produced using n equal to 100, 1000, or 10,000. The term "n/a" means that no reactions were executed. The wall-clock times needed to simulate 1,000,000 timesteps with the softwares ReaDDy (t_0) and ReaDDyMM (t_{MM}) are given. The speedup is computed from the ratio t_0/t_{MM}.

Fig. 3 compares ReaDDy's reaction kinetics with the ReaDDy reference (*yellow*) as a function of the timestep. For the system simulated here, reaction timesteps of $n\tau \leq 0.1$ μs performed well, while a reaction timestep of $n\tau = 1$ μs led to systematic errors in the kinetics. Fig. 3 shows the increase of systematic simulation error when n is set so large that the reaction timestep is on the order of the inverse fastest reaction rate. Therefore we suggest that the reaction timestep is chosen one order-of-magnitude smaller than the inverse of the fastest microscopic reaction rate, i.e., $n\tau \leq 0.1 \times (k)^{-1}$ or, equivalently, $n \leq 0.1 \times (k\tau)^{-1}$.

The diffusion timestep τ has to be chosen carefully in order to keep the Brownian dynamics integration error small. Because the integrator uses a local linearization of the nonlinear potential, it is only a good approximation when the timestep is small enough compared to the curvature of the potential. Thus, stiffer potentials require shorter timesteps (4). E.g., for a harmonic repulsion with force constant $k_{pot} = 10$ kJ mol^{-1} nm^{-2} between two particles with collision radius $r_c = 3$ nm and diffusion constant $D = 143.1$ μm^2/s, a valid timestep is $\tau = 0.1$ ns. In the future we plan to automate the choice of the timestep based on the interaction potentials employed.

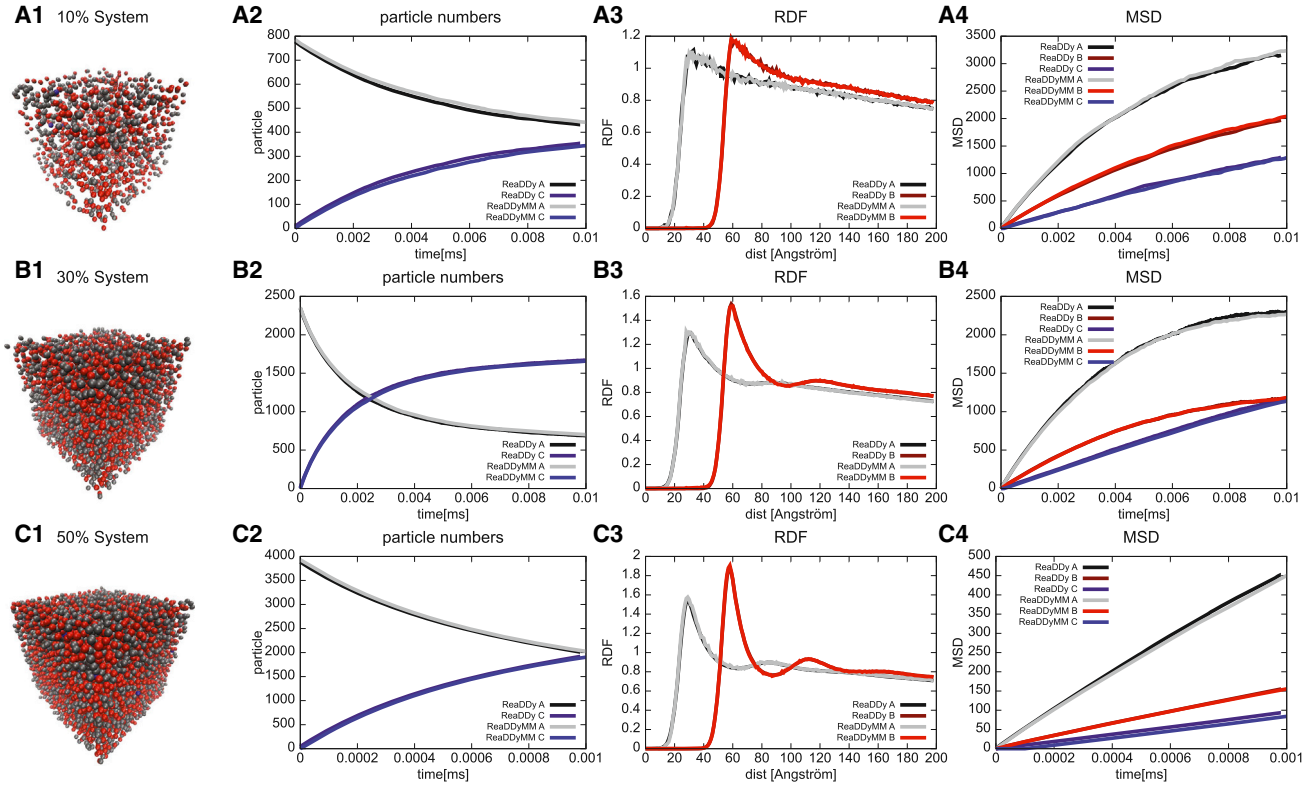

FIGURE 2 Simulation results for the reversible reaction $A + B \leftrightharpoons C$ with particles enclosed in a cube of 100-nm edge length. Particle numbers are varied to obtain 10, 30, and 50% occupied volume density in rows 1, 2, and 3, respectively. The particle and reaction dynamics timesteps were set to 0.1 ns and 0.1 μm, respectively. The first column visualizes the simulated system using VMD (9). Columns 2–4 compare different system statistics between the CPU reference implementation of ReaDDy and the CPU/GPU hybrid implementation ReaDDyMM. Column 2 shows the time evolution of particle types A and C (B is redundant). Column 3 reports the radial distribution functions of particles A and B. Column 4 shows the mean-square displacements of all particle types over time. All system statistics are accurately reproduced with ReaDDyMM. To see this figure in color, go online.

Table 1 reports the performance of ReaDDyMM using a Tesla K20c GPU (NVIDIA, Santa Clara, CA) with a Xeon CPU E5-2609, 2.40 GHz (Intel, Mountain View, CA) in comparison with the ReaDDy reference implementation running on a single Core2 Quad CPU Q6600, 2.40 GHz (Intel).

Absolute times are given for 1,000,000 dynamics simulation timesteps. Note that due to the short-ranged nature of ReaDDy, simulation times are proportional to the number of particles, N, when a fixed particle density ρ is given. In Table 1, N changes as a result of ρ, and therefore a superlinear increase of the simulation time is seen.

Systems with higher numbers of particles use the GPU more efficiently and thus benefit from a high speedup rate compared to CPU-ReaDDy. For the system with 4733 particles (50% density), the speedups are given for different choices of n, i.e., of the reaction timestep. It is seen that increasing n increases the speedup. However when using the large reaction timestep of 10^6/s with the fast reaction parameters (10^6/s association rate), this comes at the price of losing accuracy in the reaction kinetics. The maximum speedup for this system is ~115, which was obtained by switching off the reactions and executing all computations on the GPU, thus avoiding the GPU-CPU communication bottleneck.

How these wall-clock runtimes per 1,000,000 steps translate into simulated timescales strongly depends on the system setup. A timestep of 0.1 ns as used above is needed for small soluble molecules such as ATP. Typical proteins can be simulated with timesteps of ~1 ns (4), such that a coarse-grained simulation of a bacterial cytosol segment (using $\rho = 50\%$, $N = 4733$, $n = 1000$, and $\tau = 1$ ns) would require 3 h, 20 min/ms simulation time using 4733 particles, or approximately three days per ms simulation time using 100,000 particles (for comparison, *Escherichia coli* has ~3,000,000 protein copies in total).

Many signal-transduction processes, such as phototransduction or neurotransmission, are governed by slowly diffusing membrane or membrane-associated proteins permitting timesteps of ~10 ns (5). The protein volume density is low in such simulations; using $\rho = 10\%$, $N = 1000$, $n = 1000$, and $\tau = 10$ ns would require ~6 h for 1 s of simulation time. A full retinal rod cell disk membrane (~30,000 proteins) could be simulated for 1 s in approximately one week.

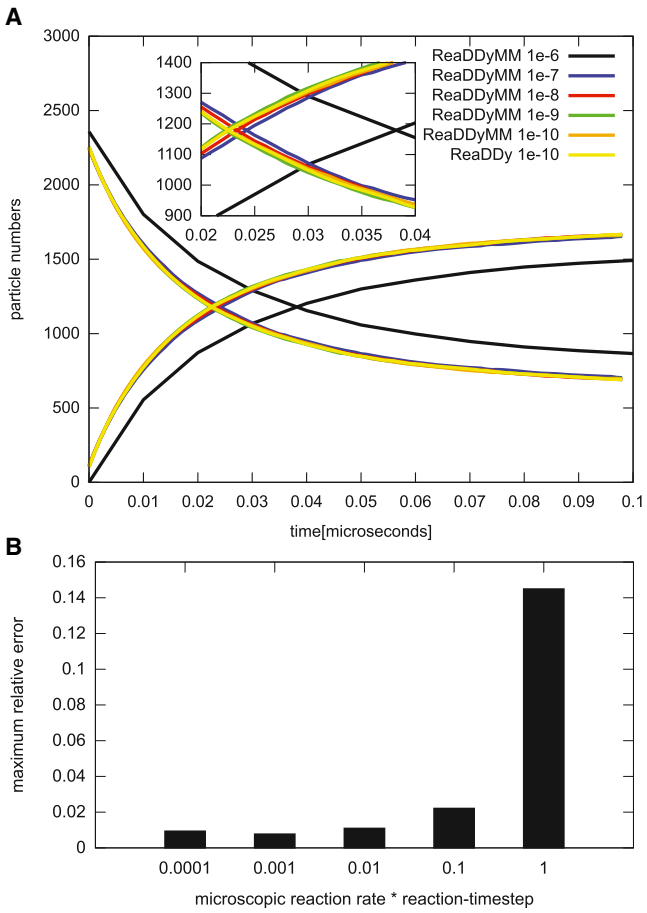

FIGURE 3 Reaction kinetics integration error as a function of the reaction timestep in a $A + B \leftrightharpoons C$ reaction with $r_{A+B\to C} = 10^6\ \mathrm{s}^{-1}$ per encounter complex. Particle motion is integrated at 0.1 ns timesteps, while the reaction timestep is varied. (*A*) Number of particles *A* and *B* (*upper curve*) and *C* (*lower curve*) over time. (*B*) Maximum relative error of the *C*-population, defined as $[C(t)] - [C_{\mathrm{ref}}(t)]/[C_{\mathrm{ref}}(\infty)]$. With a reaction timestep 10-fold faster than the inverse fastest reaction rate, the reaction kinetics are still accurately modeled, but large errors are obtained when the reaction timestep approaches the inverse of the fastest microscopic reaction rate. To see this figure in color, go online.

DISCUSSION

The coupling of the particle-base reaction-diffusion simulation package ReaDDy with the multiplatform MD package OpenMM to ReaDDyMM has resulted in a fast GPU implementation of a particle-based reaction-diffusion solver with particle interaction forces. As a result, biologically important signaling pathways that take place in seconds (e.g., neurotransmission or phototransduction) can now be simulated by explicitly resolving all involved biomolecules in space and time in a realistic cellular architecture.

This implementation exploits the fact that due to typical reaction rates in biological applications, the reaction kinetics can be integrated at a larger timestep than the timestep needed to propagate the particle dynamics. When such a timescale separation is not present, an efficient GPU implementation will require us to implement a significant part of the reaction-kinetics solver on the GPU. ReaDDy's software design allows for such a setup, and we will pursue this in the future.

We hope that ReaDDy and ReaDDyMM will be useful for the community. The code is hosted using the BSD license at https://github.com/readdy. Contributions are welcome.

ACKNOWLEDGMENTS

We kindly acknowledge funding by the German Science Foundation (Deutsche Forschungsgemeinschaft (DFG)) through SFB grant Nos. 740 and 958, as well as the European Commission through the European Research Council Starting Grant "pcCell".

REFERENCES

1. van Zon, J. S. J., and P. R. P. ten Wolde. 2005. Simulating biochemical networks at the particle level and in time and space: Green's function reaction dynamics. *Phys. Rev. Lett.* 94:128103.
2. Ridgway, D., G. Broderick, and M. J. Ellison. 2006. Accommodating space, time and randomness in network simulation. *Curr. Opin. Biotechnol.* 17:493–498.
3. Roberts, E., J. E. Stone, and Z. Luthey-Schulten. 2013. Lattice microbes: high-performance stochastic simulation method for the reaction-diffusion master equation. *J. Comput. Chem.* 34:245–255.
4. Schöneberg, J., and F. Noé. 2013. ReaDDy—a software for particle-based reaction-diffusion dynamics in crowded cellular environments. *PLoS ONE.* 8:e74261.
5. Schöneberg, J., M. Heck, …, F. Noé. 2014. Explicit spatiotemporal simulation of receptor-G protein coupling in rod cell disk membranes. *Biophys. J.* 107:1042–1053.
6. Friedrichs, M. S., P. Eastman, …, V. S. Pande. 2009. Accelerating molecular dynamic simulation on graphics processing units. *J. Comput. Chem.* 30:864–872.
7. Eastman, P., M. S. Friedrichs, …, V. S. Pande. 2013. OpenMM 4: a reusable, extensible, hardware independent library for high performance molecular simulation. *J. Chem. Theory Comput.* 9:461–469.
8. Schöneberg, J., A. Ullrich, and F. Noé. 2014. Simulation tools for particle-based reaction-diffusion dynamics in continuous space. *BMC Biophys.* 7:11.
9. Humphrey, W., A. Dalke, and K. Schulten. 1996. VMD: visual molecular dynamics. *J. Mol. Graph.* 14:27–38.