jh250081

End-to-End Differentiable Fluid-Particle Simulations

John J. Molina (Kyoto University)

Abstract

We have developed a fluid-particle simulation code in JAX/Python that is composable and end-to-end differentiable. The simulator is based on the Smooth Profile Method (SPM), which solves for the long-range many-body hydrodynamic interactions in complex particle dispersions. The resulting code leverages JAX's just-in-time (JIT) compilation and automatic differentiation (AD) support to efficiently solve forward/inverse flow problems. The same code can be run on CPUs, GPUs, or TPUs, in single or double precision, for 2D or 3D systems. We obtain excellent agreement with the reference C++ implementation, with comparable runtime on multi-core CPU systems; while GPUs offer order-of-magnitude speed increases. Furthermore, the code can be directly incorporated into existing Machine-Learning frameworks, e.g., to learn neural-network interaction potentials or constitutive relations.

1 Basic Information

1.1 Collaborating JHPCN centers

- The University of Tokyo
- Kyoto University

1.2 Theme Area

- Large-scale computational science area

1.3 Project Members and Their Roles

- J. J. Molina: Project management and model development
- T. Taniguchi: Project management and model development
- R. Yamamoto: Fluid dynamics theory
- M. S. Turner: Soft matter theory
- T. Sato: Multi-scale modeling
- S. K. Schnyder: Optimal control
- D. Mayank: Multi-scale modeling
- S. Miyamoto: Multi-scale modeling
- M. P. Lynch: Optimal control
- S. Turley: Optimal control
- F. Daiki: Flow inference
- R. Akashi: Flow optimization
- L. Kershner: Active Janus particles

2 Purpose and Significance of the Research

Particle dispersion systems, which refer to particles dispersed in host fluids, are widespread in the physical sciences, and are indispensable for many industrial and engineering applications. These systems are characterized by the long-range, non-linear, and many-body hydrodynamic interactions (HI) between the particles. To understand / control the macroscopic properties of such

systems it is necessary to properly account for the HI. Unfortunately, theoretical descriptions are limited to idealized systems (e.g., one or two particles with simple boundary conditions), which leaves computer simulations as the method of choice.

Among the many simulation methods that have been developed, i.e., Stokesian Dynamics, Lattice Boltzmann, Multi-Particle Collision Dynamics, Smooth Particle Hydrodynamics, we have chosen to use the Smooth Profile Method (SPM). The SPM directly solves the coupled Navier-Stokes and Newton-Euler equations to account for the fluid and particle dynamics. In addition, the SPM can incorporate complex / non-Newtonian fluids, arbitrary-shaped rigid particles, and it places no constraints on the Reynolds number. The principle feature of the SPM is that it replaces the sharp fluid/particle interface with a diffuse interface, allowing us to use fast pseudo-spectral methods on fixed rectangular computational grids. We have used the KAPSEL C++ code [https://kapsel-dns.com] that implements the SPM to study, among others, the sedimentation, rheology, and electrophoresis of colloidal dispersions, the dynamics of particles in compressible, viscoelastic, or multi-phase flows, as well as the collective behavior of active/swimming particles [Soft Matter, **17**, 4226, 2021].

Standard simulation codes, including KAPSEL, are ill-suited to study the inverse problems relevant for engineering or industrial applications, e.g., optimizing flow conditions, particle properties, particle-fluid affinities, and they cannot be easily incorporated within the rapidly growing Machine-Learning (ML) ecosystem. To address this limitation, we have used JAX [https://github.com/jax-ml/jax], a high-performance computing and machine learning Python library, to develop a composable and end-to-end differentiable fluid-particle simulator based on the SPM. Thanks to JAX's high-level functional programming nature, and its just-in-time (JIT) compilation and automatic differentiation (AD) capabilities, the resulting code runs efficiently in both 2D and 3D, on CPUs, GPUs, and even TPUs. This will allow us to tackle non-trivial flow optimization/inference problems, e.g., to analyze experimental data, propose novel active particles, or design improved polymer processing flows.

3 Significance as JHPCN Joint Research Project

The goal of this project is to develop a composable and end-to-end differentiable fluid/particle simulation code that can be used to solve both forward/inverse problems, with $\mathcal{O}(N^3\text{–}N^5)$ particles dispersed in complex host solvents (e.g., viscoelastic fluids, electrolyte solutions). Furthermore, we want the capability of incorporating our simulator within existing ML frameworks, e.g., to learn neural-network particle-particle interaction potentials or constitutive relations. The computational requirements for such simulations necessitate the use of large-scale high-performance CPU/GPU computing systems and expertise, as offered by the JHPCN.

4 Outline of Research Achievements until FY2024 (Only for continuous projects)

This project is not a continuous project.

5 Details of FY2025 Research Achievements

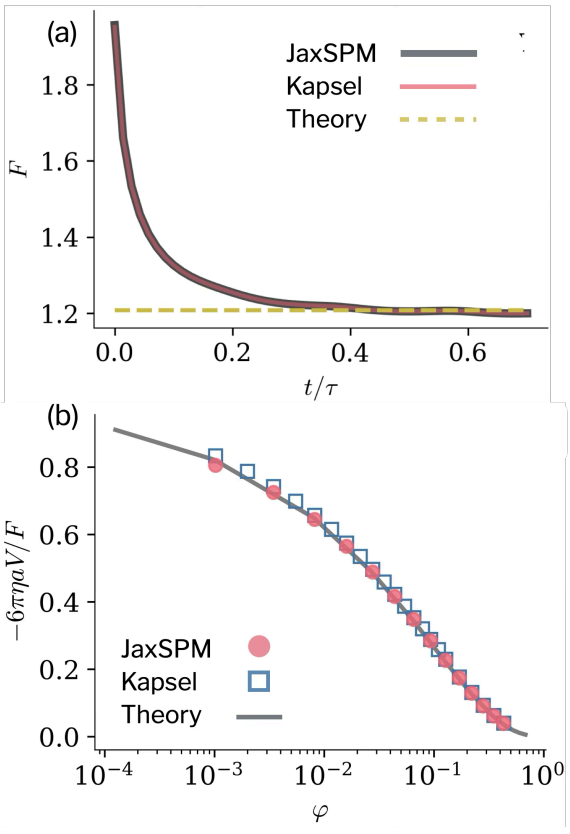

Fig. 1 Comparison between JaxSPM and Kapsel for the hydrodynamic drag force F of a single spherical particle of radius a moving at constant velocity V in a periodic system. (a) Force as a function of time for a particle of radius $a = 5\Delta$, with $\tau = a/V$. (b) Friction factor as a function of volume fraction $\varphi = 4/3\pi(a/L)^3$ obtained by varying the particle radius $a/\Delta = 4, \cdots, 30$. Theoretical values given by Hashimoto [J. Fluid. Mech. **5**, 317, 1959].

We have developed a composable and end-to-end differentiable hydrodynamic simulator (JaxSPM) that can be used to describe the motion of arbitrary-shaped rigid particles in complex host solvents. The simulator has been written using a functional programming library entirely in JAX/Python. To take advantage of JAX's AD capabilities (in both forward and backward mode), and its JIT compilation, we implemented the core of the SPM library using only the JAX native control flow primitives (e.g., `lax.scan`). To make the code easily extensible, the core library is written in terms of higher-order functions or transformers (i.e., functions that take functions as arguments and return functions), which are used to construct the actual simulator code from its constituent functions. Furthermore, we have extensively used multiple-dispatching to simplify the code base, e.g., by writing generic functions that can dispatching on the type of the field (e.g., real or complex), the dimensionality (e.g., 2D or 3D), and the state of the system (e.g., simple or complex fluid). Thanks to JAX's JIT compilation, the cost for this dispatching is paid only once, at compilation time. All of this results in an SPM library that allows us to easily change/combine the field and particle solvers, without modifying the high-lever structure of the code. In turn, this will allow us to seamlessly incorporate the different planned functionalities (e.g., complex host fluids, multi-phase flows, arbitrary shaped particles).

To validate our approach, we have compared the results obtained using JaxSPM

with those given by the reference C++ Kapsel implementation. For simplicity, we consider a system for which analytical results are known: the sedimentation of a periodic array of spheres. We performed 3D simulations for a single spherical particle (radius a, interface thickness $\xi = 2\Delta$, $\Delta = 1$ the grid spacing) moving at constant velocity ($v = 10^{-2}$), within a fluid of density $\rho = 1$ and viscosity $\eta = 1$, in a cubic simulation box of length $L = 64\Delta$, under periodic boundary conditions. The system is discretized on a rectangular grid with $64\times64\times64$ points. The results are summarized in Fig. 1, where it is seen that the JaxSPM and Kapsel results are in excellent agreement with each other, both for the transient and the steady-state values.

Next, we have implemented basic electrohydrodynamics, i.e., the ability to simulate charged particles in electrically charged fluids. In this case, we consider charged colloidal particles, whose charge is uniformly distributed over the surface, dispersed in a charged fluid. To account for the fluid charge, we include a continuum concentration field for each charged solute species. Thus, in addition to solving the Navier-Stokes and Newton-Euler equations for the fluid and particle dynamics, we must also solve for the dynamics of the charged solute, and its coupling to the fluid and colloidal particles. This is done using the Poisson-Nernst-Plank model, by solving a continuity equation with advection, diffusion, and electromigration contributions to the flux, within an electrostatic approximation, where the electric field is obtained from Gauss's law [Phys. Rev. Lett. **96**, 208302, 2006]. To test this functionality, we simulate a single negatively charged particle in the presence of a binary salt (i.e., with positively charged counterions and negatively charged co-ions), under an external field $\boldsymbol{E}_{\text{ext}} = E\hat{\boldsymbol{x}}$. The results are summarized in Fig. 2, which shows excellent agreement between the JaxSPM and Kapsel results, as well as the steady-state Poisson-Boltzmann distribution ($E = 0$). Note that,

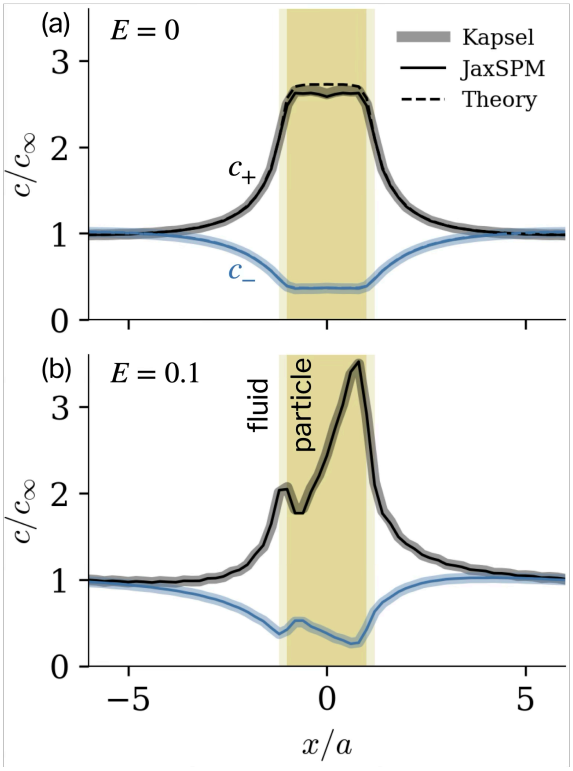

Fig. 2 Steady-state ion concentration $c_\pm$ (charge $z_\pm = \pm e$, e the unit charge), normalized by the bulk concentration c_∞, around a negatively charged colloidal particle of radius $a = 5\Delta$ and charge $Q = -100e$, in the presence of an external electric field ($\boldsymbol{E} = E_0\boldsymbol{x}$). The shaded region indicates the particle domain. Simulation parameters are taken from Kim et al. [Phys. Rev. Lett., **96**, 208302, 2006].

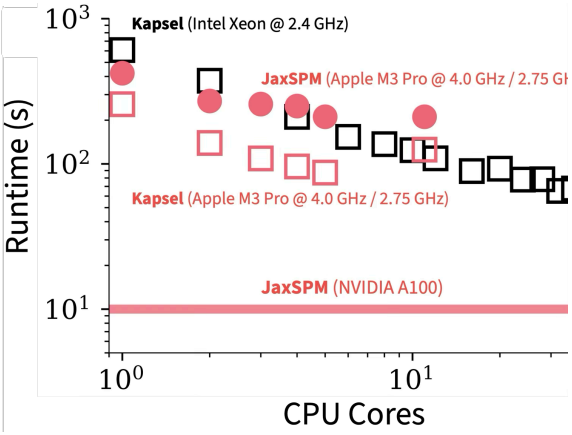

Fig. 3 Simulation runtime, in seconds, to compute the steady-state for a single sedimenting particle in 3D (see Fig. 1). The Kapsel code is only available for CPU systems, whereas JaxSPM can run on CPUs and GPUs.

for computational efficiency, the SPM uses an auxiliary concentration field $c^\star$ defined over the entire computational domain. This is the reason why the concentration is non-zero inside the particle domain. In any case, whenever the "physical" ion concentration is required, we simply multiply by the particle phase-field function.

Having established the validity of our JaxSPM simulator, we turn our focus to its computational efficiency, and the feasibility of using this code to simulate large-scale systems. As benchmark, we use the KAPSEL C++ code, which runs only on CPUs, but offers multi-core parallelization using OpenMP. We note that the KAPSEL code is highly optimized, and has seen active development for almost 20 years. In contrast, our JaxSPM code contains no such optimization. At the moment we are relying exclusive on JAX's JIT capabilities. For reference, the Kapsel code base contains $\mathcal{O}(10^4)$ lines of code, whereas JaxSPM has only $\mathcal{O}(10^2)$ lines of code (although with much more limited capabilities). Fig. 3 summarizes the runtime, as a function of the CPU cores, obtained for the particle sedimentation problem (see Fig. 1). JaxSPM is seen to be competitive with Kapsel, and at worst only $2\times$ slower. However, the main benefit comes from being able to run the same code on a GPU, e.g., with an NVIDIA A100 (Wisteria-A) the simulation is almost an order of magnitude faster than the fastest CPU run. Furthermore, depending on the simulation, we can easily gain a factor of two by running in single-precision. Such flexibility is unavailable with the reference Kapsel code.

Until now, we have only discussed the use of JaxSPM to solve "forward" flow problems. However, given the built-in AD capabilities, we can take arbitrary derivatives through the full hydrodynamic simulations! Thus, we can just as easily solve "inverse" flow problems (i.e., no modifications to the JaxSPM code are required). The basic algorithm behind this, the so-called back-propagation algorithm, is the same one used to train neural-networks. Consider an arbitrary inverse or optimization problem, for which the desired result/property is A_{target}. First, we define a suitable loss function, here given by the mean-squared error $L(\Theta) = 1/N \sum_{i=1}^{N} (A_{\text{target}} - A_{\text{sim}}(\Theta))^2$, with $A_{\text{sim}}(\Theta)$ the simulation output and Θ the simulation parameters. Second, we use JAX's JIT and AD capabilities to compute the relevant gradient function $\nabla_\Theta L$, i.e., the func-

tion that computes the gradient of the loss with respect to the simulation parameters. Finally, this grad-loss function is used as input to a gradient based optimizer, in order to find the optimal parameters $\Theta^\star = \mathrm{argmin}_\Theta L$ ($\nabla_\Theta L|_{\Theta^\star} = 0$). For this, we use the JAXopt library [https://github.com/google/jaxopt], which provides differentiable optimizers in JAX (i.e., allowing us to also differentiate with respect to the solutions of optimization problems).

To test JaxSPM on inverse problems, we have considered a simple 2D flow optimization problem. We use a pressure-driven flow setup between flat parallel plates, past a fixed particle inclusion placed at a position (x, y). The inclusion is modeled as a non-spherical particle, whose shape/contour is represented using Fourier modes (amplitude a_n, phase angle ϕ_n). Thus, the optimization parameters are the particle center-of-mass (2), and the Fourier coefficients ($2n$), for a total of $2(n+1)$parameters. First, the forward problem is solved for a given Θ, to generate the target or training data, in the form of the steady state flow velocity. Then, starting from a random initial configuration Θ_0, we solve the optimization problem of minimizing the loss function to find an optimal set of parameters $\Theta^\star$ that reproduce the target velocity profile. Here, we use the flow velocity at the channel outlet. Figure 4 shows the results in the case where only the first deformation mode is considered (i.e., $n = 1$). The optimization yields a negligible loss function $L \simeq 10^{-7}$, with the optimized velocity profile in almost perfect agreement with the target profile (indistinguishable on the scale of the plot). The optimized configuration is in close agreement with the "true" configuration; the center-of-mass position and shape are correctly reproduced, although the orientation of the particle is slightly shifted, i.e., we have found a local minimum.

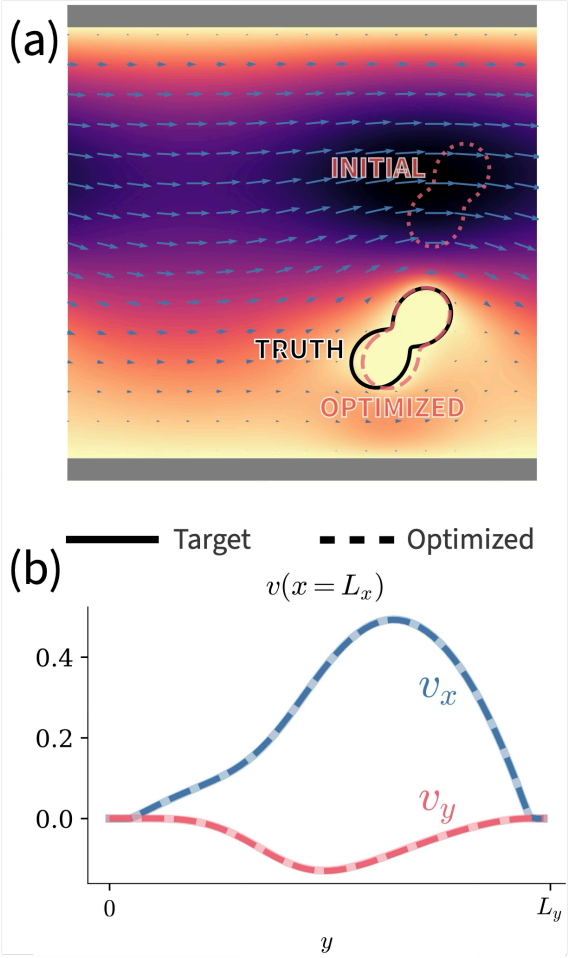

Fig. 4 Optimization results for pressure-driven flow past a particle inclusion in 2D. The center-of-mass position and shape parameters (amplitude of the deformation and phase angle) are optimized to reproduce the given velocity profile at the channel outlet ($x = L_x$). (a) Optimized steady-state velocity profile, (b) target and optimized velocity profiles.

Finally, we have extended the SPM to consider the Brownian motion of non-spherical particles within a fluctuating hydrodynamics descriptions [J. Chem. Phys, **163**, 184102, 2025], as well as explored the modeling of entangled polymer melts [Precision Chemistry, **4**, 240-252, 2026; ACS Applied Polymer Materials, **8**, 2975-2991, 2026], with the aim of incorporating complex flows into our future simulation studies.

6 Self-review of Current Progress and Future Prospects

We have successfully implemented the composable and end-to-end-differentiable hydrodynamic simulator that we intended. While we have not incorporated all the features that were originally planned for (e.g., swimming particles, phase-separating fluids), we have developed a functional SPM library that can serve as a framework for easily incorporating these features, and many more, into future simulators. Crucially, we have shown that our JAX/Python implementation is competitive with state-of-the-art and highly-optimized C++ code. More importantly, the same Python code will also run on GPU systems (achieving order of magnitude speedups compared to a CPU run). Going forward, we plan to leverage JAX's newly introduced Shardmap, which allows one to run on multi-node/multi-GPU systems, enabling large-scale simulations. This will allow us to tackle complex flow design/optimization problems relevant to engineering and industry applications (e.g., learning constitutive relations of entangled polymer melts, designing smart active particles).