Parallel-in-time Methods for Particle Simulations

Lukáš Veškrna
Technical University Munich
& Charles University (Prague)
Munich, Germany
lukas.veskrna@gmail.com

***Abstract*—This paper examines parallel-in-time integration, specifically the parareal algorithm, for particle simulations by developing a simple parallel-in-time simulator for solar system dynamics. We present experimental measurements of the simulation run times, which we then compare with expectations derived from theory. Additionally, we explore the behavior of the algorithm with different parameter settings.**

***Index Terms*—parareal, particle simulations, parallel-in-time, solar system.**

I. Introduction

Particle simulations are an important tool for understanding many different kinds of particle-based systems, such as molecular, fluid, plasma, solar system and stellar dynamics.

Simulating those systems usually requires a huge amount of computations, which can take a very long time. Some of these computations can be run in parallel in order to reduce the computation time. There has been a lot of focus in parallelizing these simulations through the spatial domain, which allows us to run bigger simulations while taking the same time.

There do, however, also exist algorithms that allow us to parallelize simulations in the time domain. This could be very useful for running smaller simulations through a very long timespan or with a very small step size.

In this article, we will focus on the fitting example of solar system dynamics, although many applications could be found also in other fields, such as molecular dynamics. [7]

To study the behavior of mostly stable planetary orbits, long simulations are required. Thus, parallel-in-time algorithms provide a promising approach. [2]

In particle simulations, we are often under the hood numerically solving an ODE in the form:

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = f(t, u(t))$$

where f is the function describing our problem, $u(t)$ is the state of the system at time t and we are given an initial condition $u(t_0) = u_0$ (this type of problem is often called an initial value problem).

We typically solve this problem using a numerical method, that does a step in time of size Δt and estimates $u(t_0 + \Delta t)$. We then use this result to advance the computation further in time by taking more such steps.[1]

Thus, the integration is the part of the computation which usually is dependent on some previous time step and thus can't be easily parallelized. In order to make it parallelizable, multiple techniques were devised, one of them being the parareal algorithm.

II. The Parareal Algorithm

The parareal algorithm was introduced by Lions, Maday and Turinici in 2001. [1][2] It achieves parallel-in-time integration using two types of integrators. A coarse solver $\mathcal{C}$ is used first to roughly estimate the solution[3], while a fine (but potentially expensive) solver $\mathcal{F}$ is then used to further refine this estimated trajectory.

Suppose we integrate from time t_0 to T, we split this interval into N segments of size $\Delta t = \frac{T-t_0}{N}$. We denote $u_0, u_1 \dots u_N$ the states of the system in times $t_0, t_1 \dots t_N$ with $t_n = t_0 + n\Delta t$ (so $t_0 = t_0$ and $t_N = T$).[4]

The algorithm works in iterations. The 0th iteration is used to initialize the algorithm, while the subsequent iterations improve the result of the previous iteration. We use the notation u_n^i to represent u_n for iteration i.

In the 0th iteration, we use the coarse solver to compute $u_1 \dots u_N$ sequentially. For every $0 \leq n < N$ we do

$$u_{n+1}^0 = \mathcal{C}(t_n, t_n + \Delta t, u_n^0)$$

with $u_0^0 = u_0$, which is the initial state of the system.

In every subsequent iteration $i + 1$, we first calculate the results of the fine but expensive solver, $\mathcal{F}(t_n, t_n + \Delta t, u_n^i)$, for every $0 \leq n < N$ in parallel.

Then, we adjust the estimated states from the previous iteration with the following correction:

$$\begin{aligned} u_{n+1}^{i+1} = \mathcal{C}(t_n, t_n + \Delta t, u_n^{i+1}) \\ + \mathcal{F}(t_n, t_n + \Delta t, u_n^i) \\ - \mathcal{C}(t_n, t_n + \Delta t, u_n^i) \end{aligned} \quad (1)$$

This is repeated until satisfactory convergence. It has been proven for example by Gander et al. [4] that for $i \to \infty$, we will converge to $u_{0\dots N}$ such that $u_{n+1} = \mathcal{F}(t_n, t_n + \Delta t, u_n)$,

[1]One such method is the explicit Euler method, which simply approximates u at time $t + \Delta t$ by taking a step in the direction given by the value of f at the current state u at time t, so $u(t + \Delta t) \approx u(t) + \Delta t f(t, u(t))$. [10]

[2]The original paper is in French but it does include a basic description in English.

[3]In the literature, the coarse solver is often denoted $\mathcal{G}$. However, I find it more intuitive to use $\mathcal{C}$ as in the word **c**oarse.

[4]The solvers can also use their own smaller subdivisions (e.g. different step sizes) but the parareal algorithm does not interact with those subdivisions.

which is equivalent to computing the states at times $t_{0\ldots N}$ with the fine solver sequentially.

In the work by Gander et al. [4], the method has been formulated using different generalized frameworks. They also extend the time interval partitioning to arbitrary partitions, not only equally sized ones as in the original paper. [1]

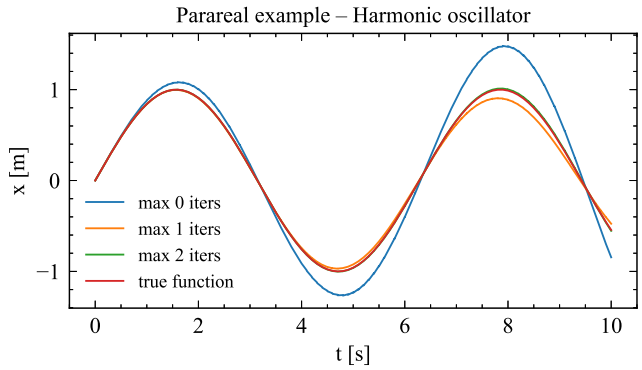

Figure 1. Parareal convergence for the example of a harmonic oscillator with $k = 1\frac{\mathrm{N}}{\mathrm{m}}$. The coarse solver is explicit Euler and the fine solver is RK4.

Stopping criteria

We assume the algorithm has converged enough when $\forall n\colon \|u_n^{i+1} - u_n^i\| < \varepsilon$. Other criteria also exist. [3]

Theoretical speed-up

A derivation of the theoretical speedup is presented in Pentland et al. [3] (Section 2.3), which we use here. It calculates the speed-up when running even the coarse solver partially in parallel, we simplify it by running $\mathcal{C}$ serially (our implementation doesn't include this optimization).

To find the theoretical speed-up of the algorithm, we compute the time required by the parareal algorithm, T_{parareal}, and the time T_{serial} taken to compute the solution by running the fine integrator serially.

If we assume running $\mathcal{F}$ over any segment $[t_n, t_{n+1}]$ takes $T_{\mathcal{F}}$ and running $\mathcal{C}$ over any segment takes $T_{\mathcal{C}}$, then computing $\mathcal{F}$ serially takes

$$T_{\text{serial}} \approx NT_{\mathcal{F}}$$

while using the parareal algorithm, with N CPUs, we have

$$T_{\text{parareal}} \approx NT_{\mathcal{C}} + \sum_{i=1}^{k} (T_{\mathcal{F}} + NT_{\mathcal{C}}) = kT_{\mathcal{F}} + (k+1)NT_{\mathcal{C}}$$

where k is the number of iterations required for convergence.

Finally, the speed-up S_{parareal} can be computed as

$$S_{\text{parareal}} = \frac{T_{\text{serial}}}{T_{\text{parareal}}} = \frac{NT_{\mathcal{F}}}{kT_{\mathcal{F}} + (k+1)NT_{\mathcal{C}}} = \left(\frac{k}{N} + (k+1)\frac{T_{\mathcal{C}}}{T_{\mathcal{F}}}\right)^{-1} \quad (2)$$

Thus, to reach a high speed-up, we want the ratio $\frac{T_{\mathcal{C}}}{T_{\mathcal{F}}}$ to be small while also keeping the number of required iterations k small. [3]

The algorithm always converges in at most N iterations, because the exact initial condition propagates through all the N segments, however, that is not useful (and takes even more time), because the algorithm has to converge at a significantly smaller number of iterations to gain any real speed-up. [3] (2.3)

III. Application to Particle Simulations

The purpose of this paper is to examine the potential in the use of the parareal method for long running simulations such as simulations of the solar system.

A. The particle simulation

We use a simple n-body particle simulation of the solar system with time complexity $\mathcal{O}(n^2)$, which we then integrate using the parareal method. This is inefficient for a large number of particles, however, our focus is on the parallel-in-time integration, not the optimizations required for a large number of particles, therefore we use a simpler simulation.

Thus, we are integrating the following equation:

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = f(t, u(t)) = f(t, (x(t), v(t))) = (v(t), a(t))$$

where

$$a_i(t) = -G\sum_j m_j \frac{x_i(t) - x_j(t)}{\|x_i(t) - x_j(t)\|_2^3}$$

where m_j is the mass of the j-th particle. Acceleration is derived solely from Newton's law of universal gravitation.

It is also possible to use a similar algorithm to the parareal algorithm using a simplified model instead of the coarse solver. This was done prior to the invention of parareal by Saha et al. [2], also in a solar system simulation, who obtained the coarse approximation by simulating only the interactions of the Sun and other planets. [2] [5]

B. Choice of the coarse and fine integrators

- We use the Velocity Verlet method as the coarse integrator for its favourable stability properties with a small cost and the classic fourth-order Runge-Kutta method (RK4) as the fine integrator for its precision. [8]
- Two symplectic methods could be used[5], however, due to the nature of the parareal algorithm, the resulting parareal integration would not necessarily be symplectic. A modified parareal algorithm that preserves the symplectic property, shown in [6], also exists, however it is beyond the scope of this paper.
- An adaptive step size fine integrator with error control would be an option, such as the Dormand-Prince RK45 method. However, since the method can have very different execution times (depending on the step size required to reach the desired accuracy), which could potentially

[5]A symplectic integrator preserves physically meaningful behavior of a Hamiltonian system in the long term. [8]

be an issue when parallelizing the code, RK4 was used for simplicity.

For completeness, the two selected integration methods are briefly described in the next paragraphs.

1) Velocity Verlet:

$$x(t+\Delta t) \approx x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$$
$$v(t+\Delta t) \approx v(t) + \frac{a(t)+a(t+\Delta t)}{2}\Delta t$$

Here $x(t)$, $v(t)$ and $a(t)$ are approximations of the corresponding values (position, velocity, acceleration) by the method at time t, Δt is the time step size. Note that only one evaluation of a per step suffices, because we can keep $a(t)$ from the last step. [9]

2) The classic fourth-order Runge-Kutta method (RK4):

$$k_1 = f(t, u(t))$$
$$k_2 = f(t + \frac{\Delta t}{2}, u(t) + \Delta t\frac{k_1}{2})$$
$$k_3 = f(t + \frac{\Delta t}{2}, u(t) + \Delta t\frac{k_2}{2})$$
$$k_4 = f(t + \Delta t, u(t) + \Delta t k_3)$$
$$u(t+\Delta t) \approx u(t) + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where $u(t)$ is the approximation of the state by the method at time t. Δt denotes the time step size. The method requires four acceleration computations per step. [10]

C. The initial conditions

The initial conditions for the simulation were generated with a python script using the astropy package. [13]

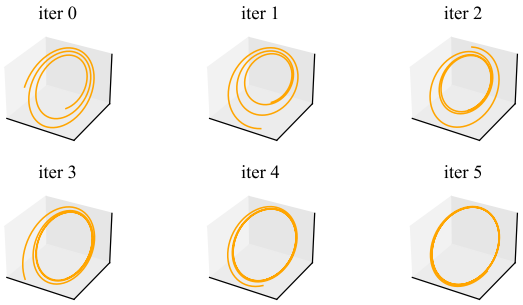

Figure 2. A more complex example to illustrate the behavior of the parareal algorithm: Path of Mercury over the time of one Earth year in a parareal integrated simulation converging over multiple iterations of the parareal algorithm. Velocity Verlet and RK4 are used with the same step size of 0.5 day.

IV. Implementation and Experiments

A. Implementation

An implementation in C++ using OpenMP for the parallelization was written.[6] In case we would want to use a substantially higher number of CPUs, we would have to use a different framework, such as MPI. However, due to the higher complexity of such implementation, a simpler approach with OpenMP was used in this project instead.

[6]The code is available at https://github.com/lesves/mol-dyn.

B. Experiments: The environment

All experiments were run on a cluster in order to get a more reproducible and comparable environment.

The experimental runs were conducted on the zia.cerit-sc.cz cluster owned by CERIT-SC/MU, a part of the MetaCentrum organization. Each computer in the cluster is equiped with two AMD EPYC 7662 (2x 64 Core) 3.31 GHz processors.

C. Experiment I: Measured run time and expected run time comparison

Multiple experiments with the same initial conditions and the same step size were conducted. A very small step size was used, so all of the runs would converge within the same number of iterations (one iteration[7]). For comparison, also a serial version (not using the parareal algorithm) was run with the same initial conditions and step size.

The resulting run times were collected in Table I. For visualizations of the simulation results see Figures 3 and 4.

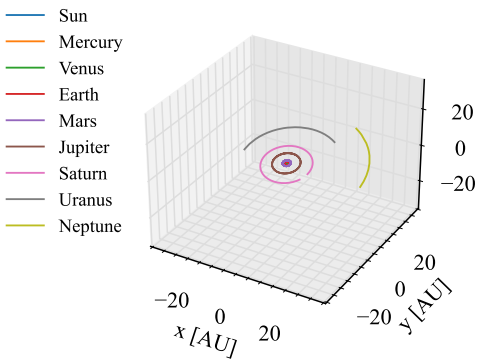

Figure 3. Trajectory computed in the simulation of 10^4 days of the solar system using parareal. For more information, see Table I.

Let's derive the expected speed-ups and run times of our program and compare them with the measured run times. To simplify the calculation, we will only estimate it using the number of function evaluations. For Velocity Verlet, we have $T_{\mathcal{C}} = 1N_{\mathcal{C}}$, where $N_{\mathcal{C}}$ is the number of steps of the coarse integrator. RK4 takes four function evaluations, so we have $T_{\mathcal{F}} = 4N_{\mathcal{F}}$. If the fine integrator does 10 times more steps than the coarse one, we have $\frac{N_{\mathcal{C}}}{N_{\mathcal{F}}} = \frac{1}{10}$. Thus, we have

$$S_{\text{parareal}} = \left(\frac{k}{N} + (k+1)\frac{1N_{\mathcal{C}}}{4N_{\mathcal{F}}}\right)^{-1} = \left(\frac{k}{N} + \frac{k+1}{40}\right)^{-1}$$

[7]This choice is explained in the next experiment (*Experiment II*), where the observations suggest that convergence within one iteration possibly has optimal run time.

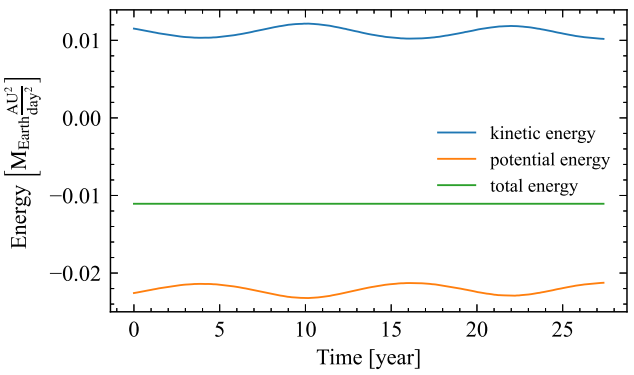

Figure 4. Conservation of energy in the simulation. Same simulation as in Figure 3. With a small enough step size, even a non-symplectic integrator like RK4 can preserve energy for some number of time steps.

When we choose the number of processors (and the number of segments), $N = 16$ and we converge within one iteration, we have

$$S_{\text{parareal}} = \left(\frac{1}{16} + \frac{1}{20}\right)^{-1} \approx 8.89$$

We can compare this with the real measured speed-up, see Table I. We observe that in reality, the program runs around 5 times faster with 16 threads. We can also compare the expected speed-up S_{parareal} for other numbers of CPUs, see Figure 5.

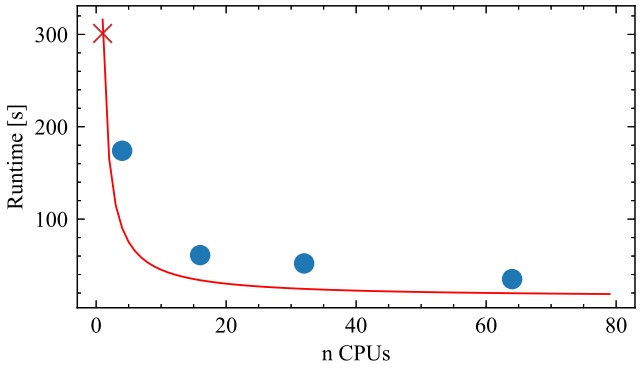

Figure 5. Comparison of the expected run times (red curve, computed from S_{parareal} and the run time of the serial algorithm (red "x" point)) and the real measured times with different numbers of CPU cores (blue points). For more information, see Table I.

While we lose some accuracy with the parareal method, this is a parameter that can be controlled, for example using the converge parameter (ε) and the coarse solver accuracy.

If we instead approximate the simulation using a similar serial simulation with a lower number of steps, we get a substantially bigger error than the parareal algorithm while not improving the run time significantly. This is also shown in Table I (as "serial (approx.)").

D. Experiment II: Run times, numbers of iterations and coarse solver steps

The second experiment shows how run times and the numbers of iterations that the program takes to converge relate to the number of steps of the coarse solver. This observation can help with deciding what is the optimal number of steps to use for the coarse solver. We are interested primarily in the run time, however the number of iterations turns out to be a very important factor in the run time.

A series of simulations with the same configuration but with different numbers of steps of the coarse solver ($N_{\mathcal{C}}$) were run and the results are shown in Figure 6. The parameter for the stopping criterion was fixed at $\varepsilon = 10^{-3}$ and we observe varying numbers of iterations required to reach this criterion.

A key feature observed in Figure 6 are the "spikes" in the run times, that occur when a change in the number of required iterations changes. After every initial downward spike, there is always an upward trend.

Presumably, this shows that the run time depends primarily on the number of iterations and only secondarily on the number of steps of the coarse solver. The upward trends correspond to improving the coarse solver while not improving the number of iterations, which does not improve the computation time.

From this, we can conjecture, that (in the case of our simulation) the optimal choice of the number of steps for the coarse solver (in the sense of the smallest run time with a fixed ε) is the smallest possible number such that parareal converges within one iteration (or in terms of step sizes, the optimal choice of the step size is the biggest possible such that the algorithm converges in one iteration).

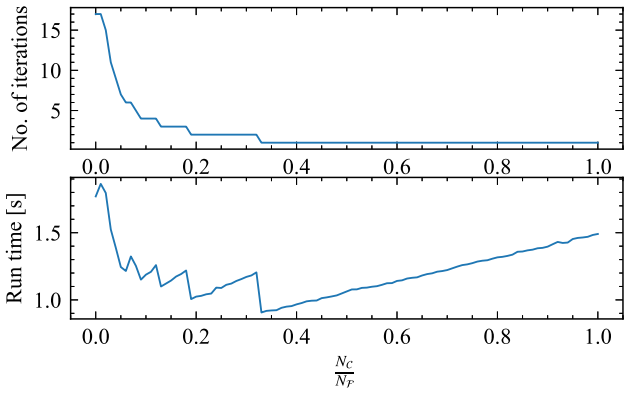

Figure 6. Run time and number of iterations depending on the ratio of the numbers of steps of the coarse and fine solvers. It corresponds to a simulation of two years with final step size $\frac{2\times 365\,\text{day}}{800\times 16} \approx 0.057$ day run on 16 cores (with $N_{\mathcal{F}} = 800$, $\mathcal{F}$ is once again RK4 and $\mathcal{C}$ is Velocity Verlet). It was averaged through 10 runs to reduce noise interfering with the small running times. The ε for convergence was chosen as $\varepsilon = 10^{-3}$.

V. Summary and Discussion

A. Speed-up

The run times of a fixed simulation with different numbers of CPU cores were measured in Experiment I and compared

Table I

Comparison of running times taken by different configurations of the simulator for a planetary simulation of 10^4 days. $N_\mathcal{C}$ and $N_\mathcal{F}$ refer to the number of steps of the coarse and the fine integrators respectively (within one segment). The fine solver step numbers have been chosen specifically to preserve the step size of 0.003125 days. This step size was selected, so all the parareal runs converge within one iteration (with $\varepsilon = 10^{-3}$ AU). This number of iterations was chosen for easier comparison between the runs and to achieve greater speed-up (see *Experiment II*). The programs were run on the machines of the zia.cerit-sc.cz cluster owned by CERIT-SC/MU, a part of the MetaCentrum organization, each equipped with two AMD EPYC 7662 (2x 64 Core) 3.31 GHz processors.

Program	n CPUs [a]	$N_\mathcal{C}$	$N_\mathcal{F}$	$\Delta t_\mathcal{F}$ [b]	Abs. err. [c]	Rel. err. [d]	CPU time	Real time
parareal	64	0.5×10^4	0.5×10^5	0.003125 day	0.0110 AU	3.60%	00:11:53	00:00:35
parareal	32	1×10^4	1×10^5	0.003125 day	0.0110 AU	3.60%	00:12:53	00:00:52
parareal	16	2×10^4	2×10^5	0.003125 day	0.0109 AU	3.60%	00:10:25	00:01:01
parareal	4	8×10^4	8×10^5	0.003125 day	0.0109 AU	3.57%	00:10:27	00:02:54
serial (base)	1	N/A	32×10^5	0.003125 day	0 AU	0%	00:05:01	00:05:01
serial (approx.) [e]	1	N/A	$0.9 \times 32 \times 10^5$	0.003472 day	6.7580 AU	256.25%	00:04:38	00:04:38

[a] The number of segments (N) is the same as the number of CPUs.

[b] The resulting step size (fine solver step size). For parareal, it can be calculated as $\frac{T-t_0}{NN_\mathcal{F}}$

[c] Maximum difference from the base serial computation across the trajectory

[d] Maximum relative error across the trajectory compared to the base serial computation

[e] A serial approximation of the base serial simulation using a bigger step size.

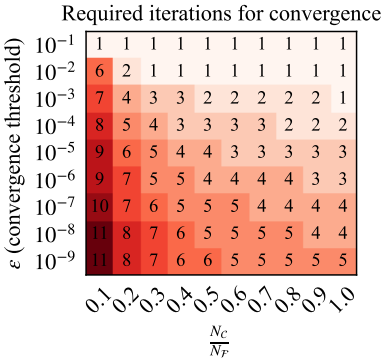

Figure 7. This bonus figure shows the numbers of iterations that the parareal algorithm with $N = 16$ requires to converge for a simulation of 365 days with the fine solver step size of $\frac{T-t_0}{NN_\mathcal{F}} = \frac{365 \text{ days}}{16 \times 100}$ for different ratios of $\frac{N_\mathcal{C}}{N_\mathcal{F}}$ and different values of the convergence criterion parameter, ε.

with the theoretical expectations. As was visualized in Figure 5, we can observe that the measured run times show a trend similar to the expected run times (corresponding to the predicted speed-ups).

The implementation is consistently slower than the estimated run times; however, many factors were not considered in the speed-up estimation, for example, in the solvers we only took into account the function evaluations. There are also other factors such as the synchronization costs of the OpenMP threads.

In conclusion, we were able to achieve noticeable speed-ups consistent with the theoretical foundation by using the parareal algorithm.

By comparing parareal with an approximating serial run using a bigger step size (see "serial (approx.)" in Table I), we have also shown that in this case, the parareal algorithm beats simply approximating the solution using a smaller step size in both accuracy and run time.

The error of the parareal algorithm can be controlled using parameters like the threshold for the convergence criterion (ε) and in some cases the coarse solver step size.

Finally, in Experiment II, we attempted to find the optimal setting (with the smallest run time and a fixed ε) of the course solver step size for our simulation. The results are presented in Figure 6. The result is a prediction of the optimal coarse solver step size for our simulation, which is the biggest possible such that the algorithm converges in one iteration.

B. Spatial vs temporal parallelization

In some situations, it is not that clear as in our case whether is it better to use spatial or temporal parallelization or both. This is beyond the scope of this article, however, there has been research in this area.

For example, Croce et al. presented a study using a simulation of unsteady Navier-Stokes equations for incompressible flow using parareal that is parallel both in time and space. They measured the speed-up with different numbers of cores assigned to the spatial and temporal parallelization and showed that they can be combined to gain higher speed-up. [11]

C. Other particle simulations: Molecular dynamics

In molecular dynamics, there are potential applications for parallel-in-time integration, such as material science. [7] Or possibly simulating protein folding, which requires extremely small step sizes (as small as femtoseconds) and great amounts of computational power. [12]

We evaluated parareal only in the example of our solar system. Those settings, however, have their own specifics, for example, planets in our solar system follow orbits with very different periods, which makes them harder to simulate in the sense that if we simulate a single orbit of Neptune, we have to simulate many orbits of Mercury (in a simulation of all the planets), which requires very small step sizes to behave correctly.

We could further improve our solar system simulations by further specializing them as was done in the work by Saha et al. [2] (a different coarse model simulating only the interactions of the Sun and other planets was used instead of a course integrator). However, this would be too specific for solar system dynamics and wound not generalize well, so this is beyond our scope.

These problems are usually not present in molecular dynamics settings and thus we could expect a more stable behavior. However, molecular dynamics present their own different challenges. This could be an interesting area to continue with this work.

Acknowledgements

I would like to thank my advisor, Samuel J. Newcome, M. Sc. and all other participants of the Seminar in Methods for Molecular Dynamics, for suggesting directions, providing valuable feedback and organizing this amazing seminar.

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] Lions, J., Maday, Y. & Turinici, G. Résolution d'EDP par un schéma en temps ≪pararéel≫. *Comptes Rendus De L'Académie Des Sciences - Series I - Mathematics*. **332**, 661-668 (2001), [Online]. Available: https://doi.org/10.1016/S0764-4442(00)01793-6

[2] Saha, P., Stadel, J. & Tremaine, S. A Parallel Integration Method for Solar System Dynamics. *The Astronomical Journal*. **114** pp. 409 (1997,7), [Online]. Available: http://dx.doi.org/10.1086/118485

[3] Pentland, K., Tamborrino, M., Sullivan, T., Buchanan, J. & Appel, L. GParareal: a time-parallel ODE solver using Gaussian process emulation. *Statistics And Computing*. 33, 23 (2022,12), [Online]. Available: https://doi.org/10.1007/s11222-022-10195-y

[4] Gander, M. & Vandewalle, S. "Analysis of the Parareal Time-Parallel Time-Integration Method." SIAM Journal On Scientific Computing. 29, 556-578 (2007), [Online]. Available: https://doi.org/10.1137/05064607X

[5] Gander, M. Is it possible to predict the far future before the near future is known accurately?. *Snapshots Of Modern Mathematics From Oberwolfach; 2019*. pp. 21 (2019), [Online]. Available: https://doi.org/10.14760/SNAP-2019-021-EN

[6] Bal, G. & Wu, Q. Symplectic Parareal. *Domain Decomposition Methods In Science And Engineering XVII*. pp. 401-408 (2008)

[7] Baffico, L., Bernard, S., Maday, Y., Turinici, G. & Zérah, G. Parallel-in-time molecular-dynamics simulations. *Physical Review E*. **66** (2002,11), [Online]. Available: http://dx.doi.org/10.1103/physreve.66.057701

[8] Leimkuhler, B., Reich, S. & Skeel, R. Integration Methods for Molecular Dynamics. *Mathematical Approaches To Biomolecular Structure And Dynamics*. pp. 161-185 (1996), [Online]. Available: https://doi.org/10.1007/978-1-4612-4066-2_10

[9] Swope, W., Andersen, H., Berens, P. & Wilson, K. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal Of Chemical Physics*. **76**, 637-649 (1982,1), [Online]. Available: https://doi.org/10.1063/1.442716

[10] Steven C. Chapra, R. Numerical Methods for Engineers. (McGraw-Hill Education,2014)

[11] Croce, R., Ruprecht, D. & Krause, R. Parallel-in-Space-and-Time Simulation of the Three-Dimensional, Unsteady Navier-Stokes Equations for Incompressible Flow. *Modeling, Simulation And Optimization Of Complex Processes - HPSC 2012*. pp. 13-23 (2014)

[12] Chodera, J., Swope, W., Pitera, J. & Dill, K. Long-Time Protein Folding Dynamics from Short-Time Molecular Dynamics Simulations. *Multiscale Modeling & Simulation*. **5**, 1214-1226 (2006), [Online]. Available: https://doi.org/10.1137/06065146X

[13] Astropy Collaboration, Price-Whelan, A., Lim, P., Earl, N., Starkman, N., Bradley, L., Shupe, D., Patil, A., Corrales, L., Brasseur, C., Nöthe, M., Donath, A., Tollerud, E., Morris, B., Ginsburg, A., Vaher, E., Weaver, B., Tocknell, J., Jamieson, W., Van Kerkwijk, M., Robitaille, T., Merry, B., Bachetti, M., Günther, H., Aldcroft, T., Alvarado-Montes, J., Archibald, A., Bódi, A., Bapat, S., Barentsen, G., Bazán, J., Biswas, M., Boquien, M., Burke, D., Cara, D., Cara, M., Conroy, K., Conseil, S., Craig, M., Cross, R., Cruz, K., D'Eugenio, F., Dencheva, N., Devillepoix, H., Dietrich, J., Eigenbrot, A., Erben, T., Ferreira, L., Foreman-Mackey, D., Fox, R., Freij, N., Garg, S., Geda, R., Glattly, L., Gondhalekar, Y., Gordon, K., Grant, D., Greenfield, P., Groener, A., Guest, S., Gurovich, S., Handberg, R., Hart, A., Hatfield-Dodds, Z., Homeier, D., Hosseinzadeh, G., Jenness, T., Jones, C., Joseph, P., Kalmbach, J., Karamehmetoglu, E., Kałuszyński, M., Kelley, M., Kern, N., Kerzendorf, W., Koch, E., Kulumani, S., Lee, A., Ly, C., Ma, Z., MacBride, C., Maljaars, J., Muna, D., Murphy, N., Norman, H., O'Steen, R., Oman, K., Pacifici, C., Pascual, S., Pascual-Granado, J., Patil, R., Perren, G., Pickering, T., Rastogi, T., Roulston, B., Ryan, D., Rykoff, E., Sabater, J., Sakurikar, P., Salgado, J., Sanghi, A., Saunders, N., Savchenko, V., Schwardt, L., Seifert-Eckert, M., Shih, A., Jain, A., Shukla, G., Sick, J., Simpson, C., Singanamalla, S., Singer, L., Singhal, J., Sinha, M., Sipőcz, B., Spitler, L., Stansby, D., Streicher, O., Šumak, J., Swinbank, J., Taranu, D., Tewary, N., Tremblay, G., Val-Borro, M., Van Kooten, S., Vasović, Z., Verma, S., De Miranda Cardoso, J., Williams, P., Wilson, T., Winkel, B., Wood-Vasey, W., Xue, R., Yoachim, P., Zhang, C., Zonca, A. & Astropy Project Contributors The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. *The Astrophysical Journal*. **935**, e167 (2022,8)