Parallel Sigma Point Trajectory Propagation with Kokkos

Scott M. Blender

Department of Mechanical, Aerospace, and Nuclear Engineering Rensselaer Polytechnic Institute, 110 8th St. Troy, NY 12180

Abstract

Effective uncertainty modeling in nonlinear dynamical systems necessitates robust numerical solutions for accurate time propagation, yet traditional methods often struggle with resource inefficiency, especially in complex, large-scale systems. This project presents a novel software solution for parallel propagation of sigma point trajectories using Kokkos, enhancing computational efficiency and producing uncertainty-aware trajectory datasets. It includes trajectory sampling, RK45-based propagation, and performance benchmarking across Serial, OpenMP, and CUDA backends. The code has been rigorously tested with Catch2 against Python-solved trajectories for a long-duration low-thrust transfer between Earth and the binary asteroid 3671 Dionysus. Additional unit tests validate the sigma-point generation via the Unscented Kalman Filter (UKF). Initial results indicate that the OpenMP backend offers the fastest propagation, with future efforts focused on expanding testing and minimizing initial CUDA overhead.

Keywords: Sigma Point Propagation, Kokkos, Runge-Kutta Integration, High-Performance Computing, GPU Acceleration

Parallel Sigma Point Propagator for Uncertainty-Aware Nonlinear Systems:
Developer's repository link: https://github.com/scottblender/computing-at-scale-2025-SB-Final-Project
Licensing provisions(please choose one): BSD 3-clause
Programming language: C++
Supplementary material: None
Nature of problem(approx. 50-250 words):
Propagating complex nonlinear systems is computationally intensive and requires substantial overhead [1]. Traditional serial implementations struggle with scalability for large-scale simulations or real-time applications.
Solution method(approx. 50-250 words):
The project utilizes the Kokkos library for parallelized sigma point trajectory propagation and employs the RK45 method for numerical integration. It supports multiple backends, including Serial, OpenMP, and CUDA, and features a modular codebase for benchmarking and extension.
Additional comments including restrictions and unusual features (approx. 50-250 words):

References

[1] Wittig, A., Di Lizia, P., Armellin, R., Makino, K., Bernelli-Zazzera, F., & Berz, M. (2015). Propagation of large uncertainty sets in orbital dynamics by automatic domain splitting. Celestial Mechanics and Dynamical Astronomy, 122(2), 239–261. https://doi.org/10.1007/s10569-015-9618-3.

Parallel Sigma Point Trajectory Propagation with Kokkos

Scott M. Blender

Department of Mechanical, Aerospace, and Nuclear Engineering Rensselaer Polytechnic Institute, 110 8th St. Troy, NY 12180

Summary

Propagating large, complex nonlinear systems is computationally intensive and poses unique challenges [1]. It is essential to incorporate uncertainty propagation into trajectory design to improve state estimation, particularly for applications like low-thrust interplanetary travel, which are susceptible to measurement and navigation errors. This further amplifies the cost and overhead required when designing highly sensitive trajectories that demand great accuracy. Given the extensive propagation required, this software aims to tackle computational inefficiencies by utilizing a C++ software suite that employs Kokkos for parallel processing of trajectories, each independent of one another.

Statement of Need

In astrodynamics applications, it is standard practice to generate a bundle of nominal trajectories a spacecraft can fly [2]. After producing the nominal trajectories, the Unscented Kalman Filter (UKF) can be used to embed uncertainty. It models state uncertainty using a fixed set of points, assuming a Gaussian distribution [3]. Employing $2n+1$ sigma points generates additional initial conditions for trajectories across a discretized time domain, where new Initial Value Problems (IVPs) can be solved with the nominal trajectory. For example, if there are 100 discrete trajectory evaluations, state variables are known from the nominal trajectory at each discrete time point. These variables can be transformed under the UKF by applying sigma points, where the sigma points are computed below in Equations (1) and (2)

$$\chi_0 = \mu \tag{1}$$

$$\chi_i = \begin{cases} \mu + \sqrt{(n+\lambda)\Sigma_i}, & \text{for } i = 1, \ldots, n, \\ \mu - \sqrt{(n+\lambda)\Sigma_i}, & \text{for } i = n+1, \ldots, 2n. \end{cases} \tag{2}$$

Here, χ_i is the i^{th} sigma point, Σ_i is the covariance, μ is the mean, n is the number of state variables, and λ is a UKF scaling parameter which governs the spread of sigma points. The mean is the discretized state estimation for a given nominal trajectory at a specific time to make this process deterministic. The covariance is a diagonal matrix governed by prescribed measurement and navigational errors. Each new Initial Value Problem (IVP) can be solved with sigma points generated for a discretized time domain. Due to the deterministic generation of sigma points and the uniqueness of each nominal trajectory, sigma point, and time step combination, these IVPs are well-suited for parallel computation. In C++, sigma point generation is performed using Cholesky decomposition to approximate the square root of the covariance matrix. An inline Runge-Kutta 4^{th} order (5) (RK45) integration method is used to propagate sigma point trajectories for each unique time step and nominal trajectory from the original bundle and discretized time domain. Using Kokkos, solving these trajectories is parallelized across the number of nominal trajectories, sigma points, and time steps, creating a highly efficient method for embedding and propagating state uncertainty into complex nonlinear systems such as interplanetary transfers. Establishing such a workflow makes it easier to predict the effects of navigational and measurement uncertainty on these systems, which is crucial for mission design in astrodynamics applications.

Related Software

FilterPy is a Python library that implements common Bayesian filters, like the UKF [4]. It supports sigma point generation and the UKF but is not a standalone trajectory propagator. Thus, it must be used in tandem with SciPy integration routines like solve_ivp, which can be slow due to overhead. This project seeks to remedy the gap in bridging sigma point generation with trajectory propagation, thereby reducing overhead, while using C++ parallelism with Kokkos.

Unit Testing and Preliminary Results

To test the accuracy of the C++ implementation, existing code from FilterPy and SciPy was used that generates expected trajectories for one nominal trajectory over a single time step as a validation method for a long-duration interplanetary transfer between Earth and 3681 Dionysus, a small binary asteroid [2]. GitHub Actions were set up for automated compilation and unit testing using Catch2 to compare the results of the C++ propagation with those obtained from Python. Figure 1 illustrates a runtime plot for three backends: Serial, OpenMP, and CUDA. CUDA performs the worst when propagating 75 nominal trajectories with 15 sigma points across varying time steps. This performance issue is primarily due to data management involving host-to-device transfers. Additionally, Table 1 summarizes the results of each backend for 1000 time steps.

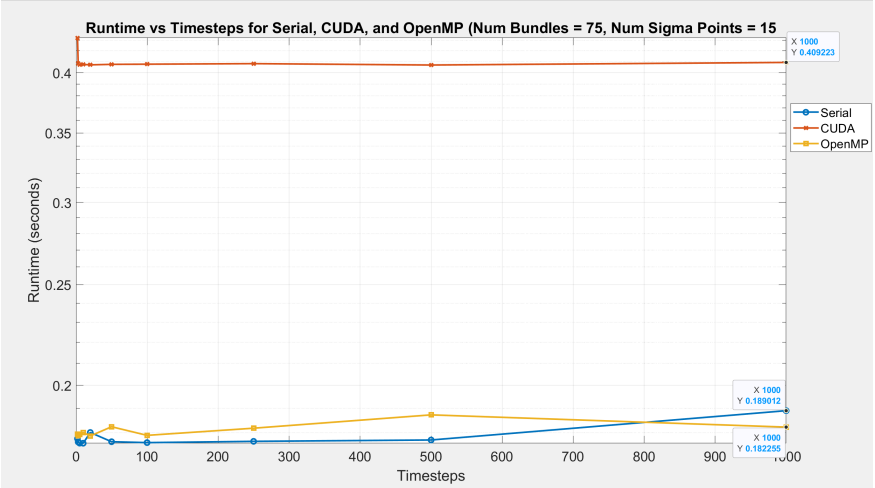

Figure 1: Runtime Comparison Plot for Serial, OpenMP, and CUDA

Table 1: Runtime comparison for 75 nominal trajectories, 15 sigma points, and 1000 timesteps.

Backend	Time (seconds)
CUDA	0.409223
OpenMP	0.182255
Serial	0.189012

Future Work

Future work will concentrate on optimizing CUDA performance by reducing expensive host-to-device data transfers. A promising approach is to refactor host-side computations, such as sigma point generation and

RK45 integration, into fully device-compatible kernels, eliminating the need for frequent memory synchronization. Additionally, implementing hierarchical parallelism using Kokkos team policies could enhance cache reuse by leveraging scratch memory, which would be particularly advantageous for intermediate computations in RK45. Benchmarking on larger problem sizes (for example, thousands of nominal trajectories or sigma points) will help evaluate CUDA performance on larger problem sizes, where its parallel throughput has the potential to surpass Serial and OpenMP backends. Furthermore, expanding the testing framework to include robustness checks under various uncertainty distributions will broaden the software's applicability in real-world astrodynamics missions.

Acknowledgements

The author would like to thank Dr. Jacob Merson and Dr. Sandeep Singh for their mentorship while working on this software.

References

[1] Alexander Wittig, Pierluigi Di Lizia, Roberto Armellin, Kyoko Makino, Franco Bernelli-Zazzera, and Martin Berz. Propagation of large uncertainty sets in orbital dynamics by automatic domain splitting. *Celestial Mechanics and Dynamical Astronomy*, 122(2):239–261, 2015. doi: 10.1007/s10569-015-9618-3.

[2] Sandeep Kumar Singh and John L. Junkins. Stochastic learning and extremal-field map based autonomous guidance of low-thrust spacecraft. *Scientific Reports*, 12:17774, 2022. doi: 10.1038/s41598-022-22730-y.

[3] Eric A. Wan and Rudolph van der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 153–158. IEEE, 2000.

[4] Roger Labbe. *Kalman and Bayesian Filters in Python*. Self-published, 2014. URL `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python`.