RACCOONS: Data Management Library for High-Performance Astrometric Reduction Pipelines

Konstantin Ryabinin[1 [0000−0002−8353−7641]], Pau Ramos[2 [0000−0002−5080−7027]], Wolfgang Löffler[1 [0009−0003−1319−5601]], and Michael Biermann[1 [0000−0002−5791−9056]]

[1] Astronomisches Rechen-Institut, Center for Astronomy of Heidelberg University, Mönchhofstr. 12–14, 69120 Heidelberg, Germany
konstantin.riabinin@uni-heidelberg.de, loeffler@ari.uni-heidelberg.de, biermann@ari.uni-heidelberg.de

[2] National Astronomical Observatory of Japan, Mitaka-shi, Tokyo 181-8588, Japan
pau.ramos@nao.ac.jp

Abstract. An astrometric reduction pipeline is a sequence of steps for calculating spatial positions and velocities of stars from their angular positions at different times on the celestial sphere as measured by telescopes. The accuracy of the results depends on the number of observations per star, because even the finest telescopes are afflicted with various noises. To reach scientifically valuable accuracy, the total number of observations undergoing reduction exceeds billions, demanding highly efficient Big Data management tools to support calculations. In this work, we introduce the RACCOONS library, which provides high-performance lock-free methods for handling immutable and mutable data. It is written in C++, has a Python binding, and implements efficient mechanisms for querying the observational data of a certain astrometric mission, calculating statistical properties of requested datasets, and acting as a large in-memory storage for mutable data by joining together the RAM of several cluster nodes, utilising the remote direct memory access. This library is currently reused in three software systems within the astrometric reduction pipeline for the Japan Astrometry Satellite Mission for Infrared Exploration (JASMINE): direct and iterative astrometric solvers, and a solution visual analytics system. At the current research and development step, we tested RACCOONS and the corresponding software on relatively small CPU clusters with 0.2 PFLOPS capacity and proved good scalability of both calculations and data treatment. The next step for future work is to deploy RACCOONS-powered software on a 2 PFLOPS cluster.

Keywords: Space Astrometry · Big Data · High-Performance Computing · Remote RAM · Lock-Free Access

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

1 Introduction

Astrometric data reduction is a Big Data problem involving the processing of billions of stellar observations made by a telescope to precisely determine stellar properties, such as celestial coordinates, proper motions, parallaxes, brightnesses, colours, etc. These parameters are combined into stellar catalogues, fostering the research of the lifecycle and structure of various objects in the Universe, including our Milky Way galaxy. An efficient and practically usable implementation of an astrometric data reduction pipeline, also known as an astrometric solver, faces a series of challenges.

First, its main bottleneck is data access, as the involved algorithms are very cache-unfriendly, exhibiting highly irregular data access patterns and requiring a large amount of random access memory (RAM).

Second, its numerical stability is not straightforward to achieve because the dynamic range of values involved in calculations by far exceeds the range of the double-precision floating-point numbers.

Third, its architectural design is non-trivial due to the trade-off between the performance and flexibility of the computational core. The flexibility is essential because the central place in the computational core is taken by a telescope calibration model, which aims to compensate for all systematic offsets in the data introduced by imperfections in the telescope's optics and electronics, as well as to describe physical effects such as relativistic light bending. This calibration model must be developed for each astrometric mission individually and includes numerous trial-and-error heuristics derived from the statistical analysis of the observational data and the residuals of the solution.

Fourth, the obtained astrometric solution requires analytical tools to verify its physical validity because, if the calibration model is not yet good enough, the revealed stellar parameters most probably do not reach the desired accuracy.

Since the European Space Agency (ESA) Hipparcos mission launched in 1989 [3], astrometry has transitioned into space, as space telescopes deliver incomparably higher accuracy and stability of observations than ground-based ones, unaffected by atmospheric noise and gravity. Still, with the ever-growing scientific demand for higher and higher accuracy, which, after the ESA Gaia mission launched in 2013, went down to 10^{-5} arcseconds in stellar position determination [4], the efficient solving of the aforementioned challenges remains pivotal for scientific success. Good practice shows that modern space astrometry missions should require data reduction pipeline redundancy by using more than one solver in parallel, each implementing different mathematical methods, thereby providing the scientists with diverse tools to detect data anomalies, cross-check different calibration models, estimate the solution uncertainties and, as a final result, enabling the veracity of the final stellar catalogue.

This paper is based on the joint contribution of the Institute for Computational Astronomy (Astronomisches Rechen-Institut, ARI) of Heidelberg University, Germany, and the National Astronomical Observatory of Japan (NAOJ) to the Japan Astrometry Satellite Mission for Infrared Exploration (JASMINE) [7]. Our two teams independently work on two different astrometric solvers, the di-

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

rect one developed at ARI and the iterative one developed at NAOJ, to facilitate the JASMINE data reduction pipeline redundancy. Despite different approaches, both solvers require high-performance data management as they intend to process the entire JASMINE dataset, which is estimated to contain nearly 10^{10} observations of more than 10^5 stars within a crowded region around the Milky Way centre. In this paper, we present the data management library RACCOONS (Rapid ACCess Operations On Numerical Solutions) that we have developed to reinforce our two solvers and our solution analysis tools.

RACCOONS tackles all four aforementioned challenges, enabling efficient operations on the data within a high-performance computing (HPC) environment. We primarily tested it on the relatively small (nearly 0.2 Peta Floating Point Operations Per Second, PFLOPS) CPU clusters `bwUniCluster` 2.0 and 3.0 available for academic research in Baden-Württemberg, Germany. Confirming good scalability, we further plan to utilise the `ATERUI III` supercomputer hosted at the Centre for Computational Astrophysics (CfCA), NAOJ, with the peak performance of nearly 2 PFLOPS.

2 Related Work

2.1 Space Astrometry Missions and Their Data Reduction Pipelines

Space astrometry is relatively young in the sense that so far, there have been only three fully successful missions, which have delivered stellar catalogues: ESA Hipparcos (1989–1993) [3], ESA Gaia (2013–2025) [4], and National Aeronautics and Space Administration (NASA) Hubble Space Telescope (HST, launched in 1990 and still operating) [1]. The upcoming missions planned for the near future are NASA Roman (to be launched around 2026/2027) [12] and Japan Aerospace Exploration Agency (JAXA) Institute of Space and Astronautical Science (ISAS) JASMINE (to be launched in 2032) [7]. While Hipparcos and Gaia are unique in continuously scanning the whole sky to provide astrometry in a global, well-characterised reference frame, JASMINE (similar to HST and Roman) is a point-and-stare mission focusing its resources on high-quality local astrometry for, in this case, the stars in the Milky Way centre.

Hippacros composes a solution from the results of two complementary data reduction pipelines: Northern Data Analysis Consortium (NDAC) pipeline [10] and Fundamental Astronomy by Space Techniques (FAST) pipeline [8]. The scanning of the celestial sphere is divided into rings containing observations made during several revolutions of the satellite. First, local astrometric solutions for these rings are calculated, and then they are iteratively combined into a global solution.

Gaia is successfully using two different types of solvers for two different purposes: Astrometric Global Iterative Solution (AGIS) [9], a block-iterative solver for the whole mission, and One Day Astrometric Solution (ODAS) [11], a direct solver for the daily control of satellite health. Both solvers reuse the so-called GaiaTools, a set of libraries for data access and basic calculations, written in Java.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

For JASMINE, we have decided to develop the software from scratch, following state-of-the-art programming techniques and modern best practices to gain maximal performance on HPC systems. The mathematical and architectural principles of AGIS and ODAS built the foundation for the two JASMINE solvers: JAXBIS (JASMINE Astrometric fleXible Block Iterative Solver), being developed at NAOJ, and AJAS (ARI JASMINE Astrometric Solver) [17], being developed at Heidelberg University. Both JASMINE solvers are designed to handle full mission data, as the relatively small scale of the mission allows this.

2.2 JASMINE Astrometric Problem

The detailed mathematical formulation of the JASMINE astrometric problem is given in [17]. Conceptually, it can be described by the following equation:

$$\mathcal{D}\boldsymbol{x} = \boldsymbol{o} - \boldsymbol{c}, \tag{1}$$

where $\mathcal{D}$ is the Jacobian matrix of the observations' model function; $\boldsymbol{x}$ is the vector of unknowns, including stellar positions, proper motions, and parallaxes, as well as the satellite's attitude and the telescope's calibration terms; $\boldsymbol{o}$ is the vector of real observations obtained by the telescope; $\boldsymbol{c}$ is the vector of modelled observations.

The equation system 1 is overdetermined and contains inconsistencies due to the inevitable irreducible noise introduced by various random factors like cosmic radiation, thermoelectrical effects, etc. So, Eq. 1 has no unique solution, and according to [5], the best fit is:

$$\boldsymbol{x}_{\text{best fit}} = (\mathcal{D}^\top \mathcal{D})^+ \cdot \mathcal{D}^\top (\boldsymbol{o} - \boldsymbol{c}), \tag{2}$$

where $(\mathcal{D}^\top \mathcal{D})^+$ is the pseudo-inverse normal matrix of system 1.

Despite a simple formulation, Eq. 2 poses a Big Data problem, as the size of $\mathcal{D}$ is in the order of $10^{10} \times 10^7$. Mathematically, Eq. 2 can be solved either directly, by calculating $(\mathcal{D}^\top \mathcal{D})^+$ in one step, or iteratively, gradually converging to the solution through a sequence of approximations.

2.3 AJAS

AJAS is written in C++ and implements the direct method of solving Eq. 2, which is comprehensively described in [17]. The advantage over an iterative method is the absence of convergence issues and the ability to straightforwardly estimate the uncertainties of the stellar parameters by extracting them from the diagonal elements of the pseudo-inverse $(\mathcal{D}^\top \mathcal{D})^+$. This comes at the cost of a much higher computational complexity of calculating the pseudo-inverse. The corresponding procedure involves forward elimination of the unknowns related to the fast-changing calibration, so that the actual matrix $\mathcal{M}$ to be inverted is within $10^6 \times 10^6$. The pseudo-inverse of the square matrix $\mathcal{M}$ is calculated by the singular value decomposition:

$$\mathcal{M}^+ = \mathcal{Z}\mathcal{E}^{-1}\mathcal{Z}^\top, \tag{3}$$

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

where $\mathcal{Z}$ is the square orthogonal matrix of eigenvectors, and $\mathcal{E}$ is the diagonal matrix of eigenvalues of $\mathcal{M}$.

We use the exascale-ready library EigenExa [18] to solve the eigenproblem and ScaLAPACK to perform the matrix-matrix multiplication. For the initial building of $\mathcal{M}$, we developed hand-crafted algorithms relying on vectorisation, multithreading, and MPI-powered multiprocessing [17].

2.4 JAXBIS

JAXBIS is written in Python and implements an iterative method of solving Eq. 2, following the philosophy of Gaia [9]. For those cases in which a direct solution is not feasible due to the limitations in memory of current HPC systems, an iterative solution can succeed by breaking down the large system of equations into smaller, more tractable problems. This is done by removing the correlations between the different types of observations' model parameters and then grouping the parameters of the observations' model (i.e. the columns of $\mathcal{D}$) by type into independent blocks such that each block can then be separated neatly into independent sets of observations (the rows of $\mathcal{D}$). The removed correlations are indirectly carried over by updating the residuals and recomputing the Jacobian after solving each block. This is a way to capitalise on the sparsity of the typical Jacobian matrix in an astrometric problem of this kind. In principle, this approach allows us to solve arbitrarily large problems but only by sacrificing interpretability and requiring careful control of the convergence.

Python is the chosen language as it is popular among modern astronomers, which ensures that JAXBIS will remain maintainable and easily exportable to other groups working on different missions. However, the heavy computations are entrusted to JAX [2] and LAPACK. LAPACK, implemented in Fortran, is accessed through SciPy and is used to efficiently solve the smaller problems via Least Squares using a QR decomposition of the partial Jacobian. JAX, however, is the main novelty of this software. It is mostly used to compute the Jacobians using auto-differentiation, thus bypassing the need to hardcode the analytical derivatives of our model and allowing us to retain flexibility without sacrificing performance or numerical accuracy. The observations' model function itself is implemented in Python in a Just-in-Time compilable manner, allowing JAX to trace it and calculate its derivatives. All aspects related to data management are handled by the RACCOONS library.

2.5 LORIS

LORIS (Layered Online Reduction Inspection System) is a Web application for analysing the solution and making the JASMINE data reduction pipeline tracktable [15]. It relies on the SciVi visual data mining platform [14], which provides a basic framework for building the scientific visualisation software upon. The LORIS server is written in Python, and the client in JavaScript, utilising HTML5, CSS3, and WebGL. LORIS provides an extensible set of statistical measures and plots that cover different aspects of the solution, allowing for the

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

inspection of its validity, identification of data anomalies, and exploration of ways to improve the solver, especially the telescope calibration model.

LORIS is designed as an it-situ visual analytics tool, which means its server runs on the same HPC system as the solver and has, therefore, direct access to the solution, avoiding expensive data copying operations.

3 RACCOONS Architecture

The RACCOONS library has grown out of the AJAS code, first being just a set of data management modules within the astrometric solver developed at ARI in 2024 [17]. As the JASMINE project progressed in 2025, NAOJ JAXBIS reached its performance limits due to a data access bottleneck. Since at that moment AJAS had already demonstrated high performance and scalability, we decided to reuse its corresponding routines, independent of the solving method, in JAXBIS to avoid code repetition and algorithm reinvention. For that purpose, we refactored the relevant modules into a standalone solver-agnostic library. The next natural application of RACCOONS is the solution analysis system LORIS, which requires efficient access to both the input and output data of the solvers to enable comprehensive solution inspection.

For maintaining the high performance, RACCOONS is written in C++, and for interfacing with JAXBIS and LORIS, it has a Python binding powered by pybind11 [6]. A schematic view of the RACCOONS architecture is shown in Fig. 1.

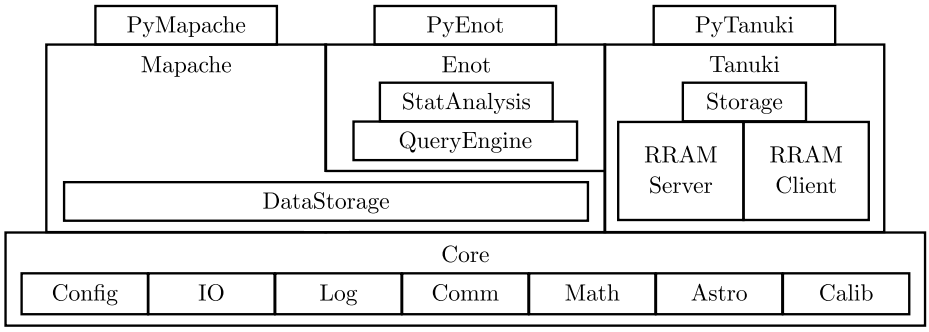

Fig. 1: RACCOONS architectural schema.

3.1 RACCOONS Core

The Core contains a set of generic modules:

1. The Config module includes the data structures and methods for configuring the RACCOONS runtime functioning and versioning.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

2. The IO (Input/Output) module provides basic functions for reading, writing, and mapping files, managing file headers and files with nested structure, as well as working with memory blocks and buffers.
3. The Log module centralises all the journaling, introducing debug, informational, warning, and error log levels, as well as managing the abnormal terminations in case of fatal errors.
4. The Comm module is a wrapper over MPI communication functions, providing individual and collective sending/receiving calls, synchronisation barriers, and process grid management, including process grouping functionality.
5. The Math module provides various mathematical functions, for example, time and angular units conversion, along with basic linear algebra primitives like vectors, rotation matrices, and quaternions.
6. The Astro module ties RACCOONS to the astrometry application domain, introducing data structures to represent astrometric missions, exposures (photos taken by the telescope), snapshots (fragments of exposures taken by individual detectors of the telescope), stars and their individual observations, as well as a generic structure of an astrometric solution.
7. The Calib module is an entry point for a telescope calibration model. The calibration model is an essential part of the astrometric solver, so it has to be plugged into RACCOONS from outside according to the solver it works with. A natural way to organise this would be a pure virtual interface with polymorphic functions. However, during the solving process, these functions are called billions of times, so we decided not to involve polymorphism here due to the function lookup overhead and to plug in the appropriate implementation statically at compile time. It ensures maximal performance but prevents reusing the library in a binary form. Improving this concept is a subject of future research and experiments.

3.2 RACCOONS Mapache

Mapache is in charge of writing and reading the input data of observations, observed stars, satellite attitude, orbital information, ephemerides, etc. It is used by all three JASMINE data reduction pipeline systems: JAXBIS, AJAS, and LORIS. Its internal module DataStorage relies on Apache Arrow to handle input files in Parquet format. Currently, DataStorage assumes Parquet schemas that meet JASMINE specifications, but thanks to modularity, they can be easily adapted for other missions. The reading is based on chunks and can be started at any point in the files, allowing for parallelisation across multiple workers of the solver. By reading, optional transformations can be applied, for example, calibration of observations according to the current calibration model and calibration parameters, update of stellar parameters and attitude according to the previously calculated astrometric solution, etc. Mapache is an interface module and therefore has a Python binding, PyMapache, providing a NumPy-compatible API.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

3.3 RACCOONS Enot

Enot, along with its Python binding PyEnot, provides methods for statistical analysis of the input data and the solution. LORIS uses it to build the initial report of the solution and then to execute the on-demand queries in an interactive mode.

Internally, Enot relies on the Mapache reading functions, and upon them, it builds QueryEngine. This module implements an engine for filtering and aggregative queries. Filtering queries allow selecting different subsets of the input data by given criteria, for example, a subset of observations belonging to a given subset of stars or to a given timespan, selecting a group of stars with given properties, etc. Aggregative queries are map-reduce operations, for example, counting observations per star, composing histograms of observational data based on different binning criteria, calculating statistical moments of certain subsets, etc.

QueryEngine provides a skeleton for the filtering and aggregative queries, each having at most three methods: Condition (to classify a data entry into a group), Selection (to extract and potentially transform a particular subset of values from the data entry), and Reduction (to define a reduction function on the extracted values). These functions are then called for each entry of the data during the query execution, which means billions of calls. Like in the Calib module, we refrain from using polymorphism. Instead, we stick with function pointers assigned to a query instance. To expose this feature to the Python binding, we use LLVM to compile the functions defined in Python at runtime, minimising the execution overhead.

The queries are executed in parallel, utilising multithreading. Currently, no inter-process communication is assumed, because the analysis is not distributed over the cluster, but instead is executed on a single node, not to overload a cluster with a task running in interactive mode. Also, the queries can be batched and chained, so that the number of data reading operations is minimised and intermediate results are reused as much as possible.

The StatAnalysis module provides presets to cover the most common analytical use cases and calculate the most instructive statistical properties of the solution, like, for example, distribution and statistical moments of solution residuals. With the deeper research of the JASMINE astrometric problem, we will add more and more methods to it.

3.4 RACCOONS Tanuki

Tanuki provides a storage for rapidly changing mutable data of large volume. Currently, it is used only by JAXBIS, due to the specifics of the iterative solving process, which implies repetitive updates of billions of solution residuals. The data structure to be handled is schematically shown in Fig. 2. The number of entries in these data equals the number of observations in the mission.

The elements o_j, $j = 0...m-1$, are m-dimensional characteristics of an observation. Physically, an observation is a light spot from a star registered by the telescope's detector. Typically, it is characterised by its centroid determined

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

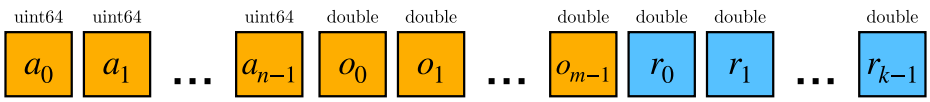

Fig. 2: Structure of a single data entry handled by Tanuki; orange depicts immutable and blue depicts mutable data.

on the data handling step preceding the data reduction. In this case, commonly $m = 4$ and o contains the 2D coordinates of the observation's centroid on the telescope's detector and corresponding 2D uncertainty of centroid determination.

The elements r_k, $k = 0...p-1$, are the p-dimensional residuals of an observation. Typically, $p = m$ and r contains the residuals of the corresponding centroid's coordinates and weights. At each step of the iterative solver, r is fully updated.

The elements a_i, $i = 0...n-1$, represent an n-dimensional index uniquely addressing each entry. Each index dimension corresponds to some physical property of the observation meaningful for the calibration model. For example, they are the index of the star the observation belongs to, the index of the exposure the observation was registered at, the index of the detector that made this observation, the index of the stellar group by colour and magnitude, and so on. n and the exact meaning of indices are determined by the calibration model and may change from run to run.

At startup, the solver splits the entire array of observations across its worker processes, determines the $\boldsymbol{a}$, $\boldsymbol{o}$, and $\boldsymbol{r}$ vectors, and submits them to the Storage module inside of Tanuki. While running the main loop, the solver constantly queries subsets of vectors $\boldsymbol{o}$ and $\boldsymbol{r}$ for certain a_i, and Tanuki provides corresponding data entries as pointers to the internal memory of Storage. Semantically, the particular query means, for example, "Get all observations and residuals for a certain subset of stars". Then, the solver makes changes to the $\boldsymbol{r}$ vectors and asks Tanuki to commit them. Query, change, and commit actions occur in parallel with a guarantee that the requested subsets do not overlap between the solver's workers. This allows Tanuki to handle the data lock-free.

The rate of queries and commits is very high, so the performance of their execution in fact determines the performance of the solver. To ensure high speed, the 3 arrays of vectors $\boldsymbol{a}$, $\boldsymbol{o}$, and $\boldsymbol{r}$ are stored separately and in different formats.

Despite each a_i being a 64-bit unsigned integer in the input, the actual size of each dimension is $d_i \ll 2^{64} - 1$. Therefore, the indices are compressed to a bitfield of length

$$l = \frac{\sum_{i=0}^{n-1} \log_2(d_i)}{64} + 1, \tag{4}$$

where l is the number of 64-bit libms.

For JASMINE, $l \leq 3$ is expected, which means $n/3$ or higher index compression ratio.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

The observations are stored in a separate file and also loaded in chunks on demand. The residuals, on the contrary, are always kept in memory because they are mutable. As their size in JASMINE is expected to be several hundred gigabytes, they certainly cannot fit into a single process memory. To solve this problem, we implemented a Remote RAM (RRAM) mechanism as a Storage backend in Tanuki. RRAM consists of client and server processes. According to the actual size of the residuals array and the available amount of RAM, several processes are excluded from the solver's process grid and assigned the RRAM server role. The residuals array is then split into x equal chunks, where x is the number of server processes. Each RRAM server spawns y threads, where y is the number of CPUs available per process. These threads then handle reading/writing requests from RRAM clients. Clients batch individual requests to minimise the number of network operations. The requests are guaranteed not to overlap; therefore, the access is lock-free.

The inter-node connection on the cluster powered by remote direct memory access (RDMA) is much faster than the write-access to the shared disk storage, so RRAM is much more efficient for mutable data than files. The heaviest part of the data access in the Tanuki Storage is the reading and matching of indices to filter out data according to the given query. On each iteration, the solver repeats the same queries over and over again because the problem configuration and work distribution never change during a single run. For this reason, the index-matching results of queries are locally cached in files saved in the local storage of cluster nodes (or on the shared disk, if local storage is not available).

Like Mapache and Enot, Tanuki also has its Python binding, PyTanuki, that provides the Storage functionality along with the basic MPI interface for the solver. All the MPI calls required by the solver, like, for example, broadcasting, scattering/gathering, and barriers, must go through Tanuki to be correctly routed in the group of RRAM client processes, so that RRAM server processes are invisible to the solver.

4 RACCOONS Evaluation

To evaluate RACCOONS, we conducted a series of tests on `bwUniCluster` 2.0 and 3.0, available for academic use at the universities of Baden-Württemberg (Germany). The tests are based on artificial data generated to mock JASMINE based on the state-of-the-art stellar catalogue that includes the JASMINE interest region of the sky around the Milky Way centre [13]. The size of the mission is mainly determined by two parameters: the number of observed stars (Λ) and the number of exposures taken by the telescope (Υ). The exposures are distributed over the interest region according to the pointing strategy of the satellite. The telescope's field of view is much smaller than the interest region, so it takes a series of pointings to fully cover it.

Each exposure takes a constant time, so Υ depends on the mission length. For JASMINE, the nominal operation time is 3 years, which gives $\Upsilon \approx 1.2 \cdot 10^6$. To unify the tests and simplify the interpretation of results, we decided to keep

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

Υ unchanged, matching the JASMINE length. So, the size of the dataset is determined by Λ, which we controlled by extracting evenly distributed subsamples of the input stellar catalogue.

Υ and Λ affect the total number of observations (L), which is the main parameter for estimating the time complexity of an astrometric data reduction pipeline through the big O notation.

Historically, the first evaluation of the RACCOONS code was a comprehensive performance analysis of AJAS [17]. At that time, RACCOONS was not yet extracted into a standalone library. However, the analysis results are still relevant, as the library extraction was a pure refactoring that introduced no changes to the implementation of underlying algorithms. For the series of gradually increasing problem sizes, AJAS revealed at least $O(L^2)$ time complexity[3], with the main contributing parts of building the reduced normal matrix $\mathcal{M}$ (39.2% of processing time on the biggest tested dataset) and solving the eigenproblem for $\mathcal{M}$ (35.2% of processing time) [17]. RACCOONS (namely, Mapache) mainly participate in the matrix building step that appears to be a processing bottleneck. However, visual analysis of the AJAS performance reported in [16] showed that this bottleneck is caused by suboptimal load balancing during parallelisation in AJAS, and not by data management methods of Mapache.

In this work, we first reveal the actual RACCOONS throughput by direct benchmarks of the library. The benchmarks utilise 2 cluster nodes of `bwUniCluster` 3.0, each containing 2 AMD EPYC™ 9454 CPUs with a total of 192 CPU cores. On these nodes, 16 processes are launched, each having 12 threads. This setup emulates real RACCOONS usage in AJAS, JAXBIS, and LORIS. For a clean benchmark, this usage is simplified to isolated sequences of calls induced by a `rank0` process, covering the entire dataset.

To benchmark Mapache, we sequentially read the entire dataset. To benchmark Enot, we query the statistical properties calculation of the entire dataset. To benchmark Tanuki, we first submit the entire dataset, read by Mapache, to the Storage submodule, where two separate copies of the same size are kept: one representing the observations and one representing the residuals. Next, we query a subset of the observations and residuals data back. In JAXBIS, this subset is used for calculations and is modified according to the results of these calculations, but in the benchmark, all the operations irrelevant to RACCOONS performance are omitted. Instead, we immediately call the method that commits the changes, even though no actual changes are made. And finally, we query the "updated" data again, which involves the query cache, because the data request is identical to the one before the commit operation. All methods are called from the `rank0` process to make the results independent from the actual parallelisation. The time of calls execution is measured to reveal the average data management rate in billions of observations per second (giga-observations per second, GOPS). The benchmark results are summarised in Table 1.

[3] Theoretically, the complexity of a direct astrometric solver is $O(\Lambda^3)$. The empirically revealed quadratic complexity of AJAS might be an artefact of a small number of dataset sizes tested. Resolving this discrepancy is a future work objective.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

Table 1: RACCOONS benchmark results. In all the test cases, $\Upsilon \approx 1.2 \cdot 10^6$; Λ is given in thousands of stars, and L is given in billions of observations.

Module	Method	Execution time (s) per test case						Rate (GOPS)
		$\Lambda = 2.5$ $L \approx 0.25$	$\Lambda = 5$ $L \approx 0.5$	$\Lambda = 10$ $L \approx 1$	$\Lambda = 30$ $L \approx 3$	$\Lambda = 50$ $L \approx 5$	$\Lambda = 70$ $L \approx 7$	
Mapache	read	24.4	48.9	95.7	294	593.2	701.9	0.01
Enot	query	33.03	33.4	36.1	250.6	539.4	790.6	0.01
Tanuki	submit	80.3	148.7	285.2	874.3	1537.5	2245.7	0.003
	query	6.4	15.6	45.05	109	216.4	224.1	0.03
	commit	0.001	0.002	0.003	0.007	0.013	0.013	364.2
	query cached	0.07	0.14	0.3	0.85	1.43	2	3.5

The RACCOONS benchmark highlights interesting trends. Mapache and Enot have almost the same performance of 0.01 GOPS. It is expected because both of them deal with data reading from input files, and their performance is capped by the speed of shared disk access. Tanuki's submit method has the lowest throughput of 0.003 GOPS, because it involves expensive shared disk writing operations along with shuffling data in RAM. Tanuki's commit operation, on the contrary, has by far the highest throughput of 364.2 GOPS, updating the data in the remote RAM. Querying the data in Tanuki works approximately 10 times faster than submitting for initial storage, and query cache speeds it up another 116 times. This means that after "warmup" (when the data are stored and all the queries are cached), Tanuki gets very efficient in providing and updating the data.

The performance of Enot can be used to estimate the performance of LORIS. As LORIS is under active development now, it is not yet fully clear how many individual Enot queries it will execute to create the full solution report. Currently, it executes just 3 queries (containing a lot of subqueries, which share the data reading results and therefore have no significant impact on the execution time), so the current throughput of LORIS is 0.003 GOPS. This means it will take about an hour to create a report for the full JASMINE containing around 10^{10} observations.

The performance of JAXBIS cannot be easily revealed from the RACCOONS benchmark, as the data management is only a part of JAXBIS operations. Therefore, we conduct separate tests. The results of these tests are still preliminary and not yet as elaborate as in the case of AJAS, due to the limited access time on `bwUniCluster` and the high demand of other academic institutions to share its usage. However, the results achieved are already instructive, revealing the scaling and time-complexity trends, as well as identifying the bottlenecks. Fig. 3 shows JAXBIS scaling with the increasing number of utilised CPU cores, while the solved astrometric problem size remains constant with $\Upsilon \approx 1.2 \cdot 10^6$ exposures, $\Lambda = 10^4$ stars, and $L = 10^9$ observations. Fig. 4 shows JAXBIS scaling with the increasing solved astrometric problem sizes, while the number of CPU cores is always equal to 768.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

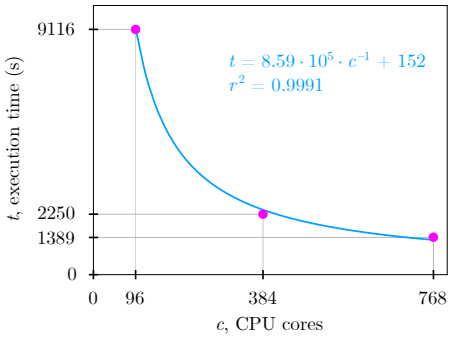

Fig. 3: JAXBIS scaling with the number of utilised CPU cores, r^2 is the goodness of the trend curve fit.

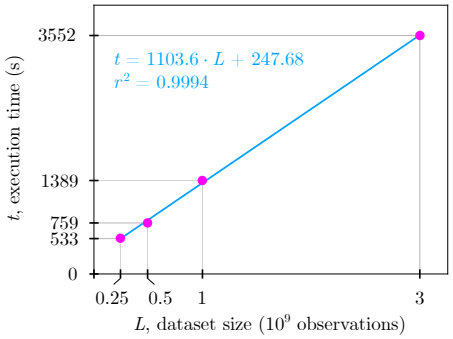

Fig. 4: JAXBIS scaling with the astrometric problem size, r^2 is the goodness of the trend line fit.

Fig. 3 shows the proportional speedup by adding the CPU cores. Fig. 4 shows that, unlike AJAS, JAXBIS time complexity is $O(L)$. This is an expected result because the number of subproblems JAXBIS solves in its block-iterative approach increases linearly from L, while the size of each subproblem stays similarly small. The revealed linear scaling also proves the efficiency of data management operations in RACCOONS, which do not bias the linear trend.

5 Conclusion

From the experiments conducted, the RACCOONS library is concluded to be a powerful and promising building block for space astrometry data reduction pipelines. It is practically useful for alleviating the data management bottlenecks in the astrometric software, like solvers and solution analysis tools, dealing with the Big Data problems on the HPC systems. To the best of our knowledge, RACCOONS is the first native data management library for astrometric data reduction pipelines driven by Big Data. It is written in C++ with lock-free direct memory access, avoiding the Just-in-Time compilation, interpretation, or garbage collection overheads that inevitably occur in state-of-the-art astrometry tools written in Java (e.g., ESA GaiaTools) or Python. However, for the usage simplicity, RACCOONS has a Python binding powered by the pybind11 library, which provides direct memory mapping from the C++ core to NumPy-compatible Python objects. We develop RACCOONS as a part of the JASMINE data reduction pipeline, but its modular architecture and generic conceptual abstraction level allow reusing it in other astrometry missions.

At the current stage of development, we settled down the RACCOONS architecture, main algorithms, and API. According to our experiments, the full JASMINE mock dataset of $\sim 10^{10}$ stellar observations can be handled by RACCOONS-powered solvers in just a few hours on a 0.2 PFLOPS HPC. But when the real mission starts, hundreds of individual runs will be needed to tune

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

the telescope's calibration model in order to reach the scientifically valuable solution accuracy. Complex distortions potentially found in the data will require increasing the complexity of the calibration model and demand more computations. Additionally, if successful, JASMINE can be extended in time (as it has happened with Gaia), and the dataset size will grow proportionally. All this inquires scalability to larger HPC systems. Therefore, the essential next step is deploying and testing RACCOONS and related software on a much bigger, 2 PFLOPS `ATERUI III` supercomputer hosted at CfCA of NAOJ.

Other future work directions are the improvement of asynchrony and fault tolerance in RACCOONS. Currently, data querying in Mapache, Enot, and Tanuki blocks the caller until the result is gathered. Making these operations asynchronous will allow parallelising input-output with calculations in the calling code. Furthermore, enhanced fault detection on top of the current basic error/integrity handling will better secure astrometric Big Data veracity.

6 Acknowledgments

This work was financially supported by the German Aerospace Agency (Deutsches Zentrum für Luft- und Raumfahrt e.V., DLR) through grant 50OD2201. The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

References

1. Benedict, G.F., McArthur, B.E., Nelan, E.P., Harrison, T.E.: Astrometry with hubble space telescope fine guidance sensors — a review. Publications of the Astronomical Society of the Pacific **129**(971), 012001 (2016). https://doi.org/10.1088/1538-3873/129/971/012001
2. Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q.: JAX: composable transformations of Python+NumPy programs (2018), http://github.com/jax-ml/jax
3. ESA: The Hipparcos and Tycho Catalogues. ESA SP-1200 (1997), https://www.cosmos.esa.int/web/hipparcos/catalogues
4. Gaia Collaboration, Prusti, T., de Bruijne, J.H.J., Brown, A.G.A., Vallenari, A., Babusiaux, C., Bailer-Jones, C.A.L., Bastian, U., Biermann, M., Evans, D.W., et al.: The Gaia mission. Astronomy & Astrophysics **595**, A1 (2016). https://doi.org/10.1051/0004-6361/201629272
5. Gauß, C.F.: Theoria Combinationis Observationum Erroribus Minimis Obnoxiae. In: Commentationes Societatis Regiae Scientiarum Gottingensis recentiores – Classis Physicae, vol. 5. Henrich Dieterich, Göttingen (1823), https://archive.org/details/theoriacombinat00gausgoog
6. Jakob, W., Rhinelander, J., Moldovan, D., et al.: pybind11 – Seamless operability between C++11 and Python (2017), https://github.com/pybind/pybind11
7. Kawata, D., Kawahara, H., Gouda, N., Secrest, N.J., Kano, R., Kataza, H., Isobe, N., Ohsawa, R., Usui, F., Yamada, Y., et al.: JASMINE: Near-infrared astrometry and time-series photometry science. Publications of the Astronomical Society of Japan **76**(3), 386–425 (2024). https://doi.org/10.1093/pasj/psae020

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41

8. Kovalevsky, J., Walter, H.G., Hering, R., Röser, S., Wielen, R., Bernstein, H.H., Lenhardt, H.: The FAST Hipparcos data reduction consortium: overview of the adopted reduction software. Astronomy & astrophysics (1992), http://articles.adsabs.harvard.edu//full/1992A%26A...258....7K/0000007.000.html
9. Lindegren, L., Lammers, U., Hobbs, D., O'Mullane, W., Bastian, U., Hernàndez, J.: The astrometric core solution for the Gaia mission. Overview of models, algorithms, and software implementation. Astronomy & Astrophysics **538**, A78 (Feb 2012). https://doi.org/10.1051/0004-6361/201117905, http://cdsads.u-strasbg.fr/abs/2012A%26A...538A..78L
10. Lindegren, L., Høg, E., van Leeuwen, F., Murray, C., Evans, D., Penston, M., Perryman, M., Petersen, C., Ramamani, N., Snijders, M.: The NDAC Hipparcos data analysis consortium - Overview of the reduction methods. Astronomy & Astrophysics **258**, 18–30 (1992), https://ui.adsabs.harvard.edu/abs/1992A%26A...258...18L/abstract
11. Löffler, W., Bastian, U., Biermann, M., Jordan, S., Brüsemeister, T., Stampa, U., Bernstein, H.H.: The One-Day Astrometric Solution for the Gaia Mission. Astronomy & Astrophysics (in preparation)
12. Perkins, J.S., Wollack, E.J., Content, D.A., Abel, J.C., Baker, J.L., Bartusek, L.M., Bolcar, M.R., Han, L.L., Harper, A.M., Kruk, J.W., et al.: The Roman Space Telescope observatory build, test, and verification status. In: Coyle, L.E., Matsuura, S., Perrin, M.D. (eds.) Space Telescopes and Instrumentation 2024: Optical, Infrared, and Millimeter Wave. vol. 13092, p. 130920R. International Society for Optics and Photonics, SPIE (2024). https://doi.org/10.1117/12.3022616
13. Ramos, P., Kawata, D., Ohsawa, R., Nishiyama, S., Sanders, J., Smith, L., Koshimoto, N., Minniti, D., Lucas, P.W.: Building the largest mock astrometric catalogue of the Milky Way centre in the near infrared for the end-to-end simulation of the JASMINE satellite. In: Ibsen, J., Chiozzi, G. (eds.) Software and Cyberinfrastructure for Astronomy VIII. vol. 13101, p. 131012T. International Society for Optics and Photonics, SPIE (2024). https://doi.org/10.1117/12.3018544
14. Ryabinin, K., Chumakov, R., Belousov, K., Kolesnik, M.: Ontology-Driven Visual Analytics Platform for Semantic Data Mining and Fuzzy Classification. Frontiers in Artificial Intelligence and Applications **358**, 1–7 (2022). https://doi.org/10.3233/FAIA220363
15. Ryabinin, K., Löffler, W., Erokhina, O., Sarras, G., Biermann, M.: Making Astrometric Solver Tractable Through In-Situ Visual Analytics. In: Paszynski, M., Barnard, A.S., Zhang, Y.J. (eds.) Computational Science – ICCS 2025 Workshops. pp. 154–167. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-97573-8_11
16. Ryabinin, K., Löffler, W., Biermann, M.: Visual Mining of Astrometric Solution: Let Numerics Meet Art. Frontiers in Artificial Intelligence and Applications **417**, 131–137 (2026). https://doi.org/10.3233/FAIA251651
17. Ryabinin, K., Sarras, G., Löffler, W., Erokhina, O., Biermann, M.: AJAS: A High Performance Direct Solver for Advancing High Precision Astrometry. Journal of Computational Science **87**, 102554 (2025). https://doi.org/10.1016/j.jocs.2025.102554
18. Sakurai, T., Futamura, Y., Imakura, A., Imamura, T.: Scalable Eigen-Analysis Engine for Large-Scale Eigenvalue Problems, pp. 37–57. Springer Singapore, Singapore (2019). https://doi.org/10.1007/978-981-13-1924-2_3

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_41