Interprocess Communication of Algebraic Data

Antony Della Vecchia
TU Berlin
Berlin, Germany, 10623
vecchia@math.tu-berlin.de

Abstract

We discuss implementation details of `OSCAR`'s serialization framework, highlighting the design decisions that allow the fine tuning of serialization methods for specific use cases. In particular, we show how the `mrdi` file format can be used for distributed computing.

1 Introduction

Serialization is the process of translating data in main memory to a format that can be stored or transmitted such that the data can be reconstructed on a different process or even machine. Although effective methods exist for general-purpose computing, serialization of algebraic data is a difficult problem, the main obstacle being the reconstruction of the correct algebraic context, which we elaborate on in Section 2. The `mrdi` file format [6], developed as part of the Mathematics Research Data Initiative (MaRDI) [9], was designed to address this problem. While the focus of [6] was set on long-term storage, here the format can be used for broader applications. Here we show examples for inter-process communication for distributed computations. Our open source implementation serves as a blueprint for other systems looking to adopt similar approaches.

`OSCAR` [5, 10] is a free and open source computer algebra system written in the `Julia` programming language [1]. `OSCAR` provides a core serialization framework and implementations for serializing (and deserializing) all core types to (and from) the `mrdi` file format. Building on tools for distributed computing in `Julia`'s standard library, `OSCAR` provides support for multiprocess methods with algebraic types. The details are discussed in Section 3.

Section 4 presents two specific use cases demonstrating significant speedups: computing components of the vanishing ideal of the fourth secant variety of $\mathbb{P}^3 \times \mathbb{P}^3 \times \mathbb{P}^3$ up to degree five [2], and computing the determinant of a matrix with entries in a univariate polynomial ring over $\mathbb{Z}$. The examples are available at `https://github.com/dmg-lab/OscarParallelExamples` .

Acknowledgements

We thank Claus Fieker, Friedemann Groh, Jereon Hanselman, Michael Joswig, Benjamin Lorenz, Bern Sturmfels and Matthias Zach, as well as three anonymous reviewers. This work was funded the Deutsche Forschungsgemeinschaft (DFG), project number 460135501, NFDI 29/1 "MaRDI – Mathematische Forschungsdateninitiative".

2 The `mrdi` File Format

The `mrdi` file format is designed for algebraic context reconstruction. For example, two polynomials can only be multiplied if the computer algebra system recognizes them as elements of the same polynomial ring, where recognition means their parent rings point to the same memory location. The key feature of the format is the ability to transfer this context between different `OSCAR` sessions and versions, with prototypes to support transfers to other computer algebra systems; e.g., see `https://github.com/JHanselman/MagmaMaRDI-JSON` for Magma [3]. The format is JSON-based and, similar to the `polymake` file format [7] and has the structure of an annotated tree with four main subtrees: `_type`, `data`, `_refs`, and `_ns`. The `_type` indicates the type of the stored object and its parameters, `_refs` stores context objects identified by UUIDs, `data` stores the actual serialized data, and the `_ns` is used for semantic separation, currently this is done by recording the system and version which serialized the file. See Figure 1 for a simple example. JSON was chosen for its human readability: files can be inspected without specialized tools, upgrade scripts can operate via pure text manipulation, and the format is free to evolve without requiring a fixed optimal representation from the outset. Such an approach has been successfully employed with the `polymake` format for over twenty years.

```
{ "_ns": {"Oscar": [ "https://github.com/oscar-system/Oscar.jl", "1.8.0" ] },
  "_type": { "name": "MPolyRingElem",
             "params": "c33bb5fb-96d7-42ed-8cd9-89b83eb7a298"},
  "data": [ [ [ 3, 0 ], "1" ], [ [ 1, 1 ], "-1" ], [ [ 0, 0 ], "1" ] ],
  "_refs": {
    "c33bb5fb-96d7-42ed-8cd9-89b83eb7a298": {
      "_type": { "name": "MPolyRing", "params": {"_type": "QQField" } },
      "data": {"symbols": [ "x", "y" ] } } } }
```

Figure 1: The bivariate polynomial $p = x^3 - xy + 1$ with coefficients $\mathbb{Q}$.

3 `OSCAR`'s Serialization Framework

We first describe how `OSCAR`'s serialization framework operates for long-term storage, by illustrating the process using the bivariate polynomial $p = x^3 - xy + 1$ over $\mathbb{Q}$ shown in Figure 1. Serialization proceeds as follows: first the `_ns` is written, then a shallow pass through p builds the `_type` subtree, determining the type `MPolyRingElem` (element of a multivariate polynomial ring) and registering the parent ring with a UUID in the `SerializerState`, and finally the terms of p are written to `data` as an array of tuples representing exponent vectors and coefficients. All registered parameters are then written to `_refs`, in this case the parent ring encoded with its coefficient ring and generator symbols. Deserialization reverses this process: `OSCAR` reads the `_type` subtree, checks whether the parent ring has already been deserialized via its UUID, deserializes it from `_refs` if not, and reconstructs p in the correct algebraic context. A `GlobalSerializerState` allows contexts to be shared across multiple files, a feature that is also necessary for multi-process methods. If compatible parameters are passed to load, the data is instead coerced into that context.

While the above suffices for long-term storage, the resulting files are quite verbose. For inter-process communication, `OSCAR` uses its `IPCSerializer`, omitting the `_ns` subtree, as each process runs the same version of `OSCAR`, and the `_refs` subtree by ensuring contexts are preloaded on each receiving process. To manage this automatically, `OSCAR` provides an `OscarWorkerPool`, a subtype of `Julia`'s `AbstractWorkerPool`, which handles context distribution via a post-order traversal of the `_type` subtree, guaranteeing each context is sent before the types that depend on it. This allows users to use functions from `Distributed.jl` such as `remotecall` and `pmap` directly with `OSCAR` types. The `OscarWorkerpool` also works with different cluster managers (e.g. `SlurmClusterManager.jl`) allowing users to run computations with over 1000 cpus. Further optimizations modify the `data` subtree and rely on `Julia`'s parametric type system and multiple dispatch: the `SerializerState` and `DeserializerState` are parameterized by the serializer type, allowing dispatch to select the encoding depending on the context and application. Figure 2 shows multiple encodings for parts of the `data` subtree that involve univariate polynomials. For long-term storage the encoding is sparse and avoids ambiguities in the order of coefficients.

```
function save_object(s::SerializerState, p::PolyRingElem)
  save_object(s, [
    (i - 1, c) for (i, c) in enumerate(coefficients(p))
      if !is_zero(c) ])
end

function save_object(s::SerializerState{IPCSerializer}, p::PolyRingElem)
  save_object(s, collect(coefficients(p)))
end
```

Figure 2: A snippet with the two encodings for univariate polynomials.

4 Use Cases

We note that small computations are unlikely to benefit from these methods, as the overhead of inter-process communication outweighs any potential gain. Determining which problems are large enough to warrant parallelization is still an active area of experimentation. The examples were choosen for their mathematical relevance, not for their size.

Computing Ideal Components from Algebraic Statistics. The main algorithm in [4] computes the kernel of a map ϕ homogeneous with respect to a $\mathbb{Z}^k$-multigrading up to a given total degree by iterating over multidegrees and constructing matrices whose rows are indexed by monomials of that multidegree. Row reducing these matrices over $\mathbb{Q}$ determines the kernel components degree by degree, avoiding a full elimination algorithm. The original `Macaulay2` implementation can compute examples with roughly 50,000 monomials and 25,000 multidegrees, but won't terminate on an example with 2,829,056 monomials over 637,440 multidegrees. The `OSCAR` implementation `components_of_kernel` developed as a part of [2] can handle this example, however the example is still too small for parallel methods, see [2] for a time comparison between the two implementations.

When examples are large enough to warrant the overhead, the row reduction steps may be handled independently by seperate processes. The authors of [2] were able to compute the components of the vanishing ideal of the fourth secant variety of the Segre variety $\mathbb{P}^3 \times \mathbb{P}^3 \times \mathbb{P}^3$ up to total degree five an example with 8,303,632 monomials over 175,616 multidegrees in under three hours using an Intel Xeon Silver 4216 with 32 processes and 150 GB of main memory, an example that would take `OSCAR` over 24 hours if computed with a single process.

Modular Determinant Computations for Real Plane Algebraic Curves. Computing discriminants of curves arising from recent work on smooth real plane algebraic curves of degree seven [8] is of interest. Our input is the matrix M of size 48×48 with entries in a univariate polynomial ring over $\mathbb{Z}$ whose determinant realizes one such discriminant. A standard approach [11] is to reduce the matrix over sufficiently many primes and then use the Chinese Remainder Theorem to find the determinant in $\mathbb{Z}[x]$. This approach computes $\det M$ in 1326 seconds on an Intel Core i7-3930K with 150 GB of main memory using a single process and in 357 seconds using six processes.

References

[1] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. Julia: A fresh approach to numerical computing. *SIAM Review 59*, 1 (2017), 65–98.

[2] Boege, T., et al. Algebraic statistics in `OSCAR`, 2026. arXiv:2601.15807.

[3] Bosma, W., Cannon, J., and Playoust, C. The `Magma` algebra system. I. The user language. *J. Symbolic Comput. 24*, 3-4 (1997), 235–265.

[4] Cummings, J., and Hollering, B. Computing implicitizations of multi-graded polynomial maps. *J. Symbolic Comput. 132* (2026), Paper No. 102459, 15.

[5] Decker, W., Eder, C., Fieker, C., Horn, M., and Joswig, M., Eds. *The Computer Algebra System `OSCAR`: Algorithms and Examples*. Springer, 2024.

[6] Della Vecchia, A., Joswig, M., and Lorenz, B. A FAIR file format for mathematical software. In *Mathematical software – ICMS 2024* (2024), K. Buzzard, A. Dickenstein, B. Eick, A. Leykin, and Y. Ren, Eds., no. 14749 in LNCS, Springer, pp. 234–244.

[7] Gawrilow, E., Hampe, S., and Joswig, M. The `polymake` XML file format. In *Mathematical software – ICMS 2016.* (2016), Berlin: Springer, pp. 403–410.

[8] Geiselmann, Z., et al. 121 patchworked curves of degree seven, 2026. arXiv:2602.06888.

[9] The MaRDI consortium. MaRDI: Mathematical Research Data Initiative Proposal, May 2022. doi:10.5281/zenodo.6552436.

[10] The `OSCAR` Team. `OSCAR` – Open Source Computer Algebra Research system, Version 1.8.0, 2024. `https://www.oscar-system.org`.

[11] von zur Gathen, J., and Gerhard, J. *Modern computer algebra*, third ed. Cambridge University Press, Cambridge, 2013.