REPORT ON OpenDreamKit DELIVERABLE D3.11

HPC enabled SAGE distribution

ALEXIS BREUST, KARIM BELABAS, JEAN-GUILLAUME DUMAS, JEROEN DEMEYER, WILLIAM B. HART, STEVE LINTON, CLÉMENT PERNET, HONGGUANG ZHU

Due on	31/08/2019 (M48)
Delivered on	01/09/2019
Lead	Université Grenoble Alpes (UGA)
Progress on and finalization of this deliverable has been tracked publicly at: https://github.com/OpenDreamKit/OpenDreamKit/issues/60	

DELIVERABLE DESCRIPTION, AS TAKEN FROM GITHUB ISSUE #60 ON 2019-09-02

- **WP3:** Component Architecture
- **Lead Institution:** Univ. Grenoble Alpes
- **Due:** 2019-08-31 (month 48)
- **Nature:** Other
- **Task:** T3.5 (#54)
- **Proposal:** p. 42
- **Report** (sources)

The primary use of computational mathematics software is to perform experimental mathematics, for example for testing a conjecture on as many as possible instances of size as large as possible. In this perspective, users seek for computational efficiency and the ability to harness the power of a variety of modern architectures. This is particularly relevant in the context of the OpenDreamKit Virtual Research Environment toolkit which is meant to reduce entry barriers by providing a uniform user experience from multicore personal computers -- a most common use case -- to high-end servers or even clusters. Hence, in the realm of this project, we use the term High Performance Computing (HPC) in a broad sense, covering all the above architectures with appropriate parallel paradigms (SIMD, multiprocessing, distributed computing, etc).

Work Package 5 has resulted in either enabling or drastically enhancing the high performance capabilities of several computational mathematics systems, namely Singular (D5.13 #111), GAP (D5.15 #113) and PARI (D5.16, #114), or of the dedicated library LinBox (D5.12 #110, D5.14 #112).

Bringing further HPC to a general purpose computational mathematics systems such as `SageMath` is particularly challenging; indeed, they need to cover a broad -- if not exhaustive -- range of features, in a high level and user friendly environment, with competitive performance. To achieve this, they are composed from the ground up as integrated systems that take advantage of existing highly tuned dedicated libraries or sub-systems such as aforementioned.

Were report here on the exploratory work carried out in Task to expose HPC capabilities of components to the end-user level of an integrated system such as `SageMath`.

Our first test bed is the `LinBox` library. Its multicore parallelism features have been successfully integrated in `Sage`, with a simple API letting the user control the desired level

of parallelism. We demonstrate the efficiency of the composition with experiments. Going beyond expectations, the outcome has been integrated in the next production release of `SageMath`, hence immediately benefiting thousands of users.

We proceed by detailing the unique challenges posed by each of the Singular, Pari, and GAP systems. The common cause is that they were created decades ago as standalone systems, represent hundreds of man-years of development, and were only recently redesigned to be usable as a parallel libraries. Some successes were nevertheless obtained in experimental setups and pathways to production are discussed (together with best practices that parallel libraries should follow).

CONTENTS

1. EXPOSING THE PARALLEL FINITE FIELD LINEAR ALGEBRA LIBRARY LINBOX

We were then more successful in exposing the parallel features of the LINBOX library, and more specifically of its kernel for finite field linear algebra, `fflas-ffpack`. By nature, a library is designed with composability in mind, and thread safety is among the required features a parallel library should provide.

1.1. Context

Finite field linear algebra is a core building block in computational mathematics. It has a wide range of applications, including number theory, group theory, combinatorics, etc. SAGEMATH relies on the FFLAS-FFPACK library for its critical linear algebra operations on prime fields of less than 23 bits and consequently also for numerous computations with multi-precision integer matrices.

The FFLAS-FFPACK library had some preliminary support for multi-core parallelism for matrix multiplication and Gaussian elimination. Instead of being tied to a specific parallel language, the library uses a Domain Specific Language, Paladin [3], to provide the library programmer with unique API for writing parallel code, which is then translated into OpenMP [5], Cilk [1], Intel-TBB [4], or XKaapi [2] directives. Beside portability and independence from a given technology, this also makes it possible to benchmark and compare how parallel runtimes perform. This is particularly important here, since many of the compute intensive routines of FFLAS-FFPACK share specificities that are often challenges for parallel runtime:

Recursion: by design, sub-cubic linear algebra algorithms are recursive and so are most routines in the library.

Heterogeneity: many exact computations must deal with data with size unknown before their actual computation. For instance rank deficiencies in Gaussian elimination may generates a block decomposition of varying dimensions and therefore computing tasks will have heterogeneous load.

Fine grained task parallelism: the combination of the two above constraints leads to consider recursive task fine-grained parallelism, such that a work-stealing engine could efficiently balance the heterogeneity. However, recursive tasks have been for a long time rather inefficient in e.g. OpenMP implementations, and the ability to handle numerous small tasks is also demanding on parallel runtimes.

1.2. Integration within SageMath

The main tasks for the exposition of the parallel routines of FFLAS-FFPACK in SageMath were the following:

(1) improving existing parallel code for Gaussian elimination and matrix multiplication in the FFLAS-FFPACK library;
(2) adding new parallel routines in FFLAS-FFPACK for the most commonly used operations in SageMath: the determinant, the echelon form, the rank, and the solution of a linear system;
(3) connecting these parallel routines in SageMath providing the user with a precise control on the number of threads allocated to the linear algebra routines.

The first two items involved 15 pull-requests, merged and released in `fflas-ffpack-2.4.3`[1]. This release was produced simultaneously with that of the two other libraries in the LinBox ecosystem: `givaro-4.1.1`[2] and `linbox-1.6.3`[3] then integrated into SageMath in tickets

- https://trac.sagemath.org/ticket/26932 and
- https://trac.sagemath.org/ticket/27444,

which will appear in release 8.9 of SageMath.

As a side note, the integration of these new releases including the contributions to D5.12 lead to a significant speed-up in the sequential computation time of finite field linear algebra in SageMath, as show in Table 1.

	$\mathbb{Z}/4\,194\,301\mathbb{Z}$		$\mathbb{Z}/251\mathbb{Z}$	
	Before	After	Before	After
Matrix product	3.61	3.57	1.59	1.5
Determinant	2.96	1.52	1.54	0.731
Echelon form	3.59	1.86	1.82	0.692
Linear system	8.9	5.13	3.7	1.79

TABLE 1. Improvement of the sequential code with `fflas-ffpack-2.4.3`. Computation time in seconds for a 4000×4000 machine over a 22 bits and a 8 bits finite field, on an Intel i7-8950 CPU.

For Item (3), we explored several options and chose rely on and extend the Singleton class `Parallelism` in SageMath. This class work as a dictionary registering the number of threads with which each component in SageMath can run in parallel.

For example, the following code requires than any linear algebra routine relying on linbox be parallelized on 16 cores.

```
sage: Parallelism().set("linbox",16)
```

[1] https://github.com/linbox-team/fflas-ffpack/releases/tag/2.4.3
[2] https://github.com/linbox-team/givaro/releases/tag/4.1.1
[3] https://github.com/linbox-team/linbox/releases/tag/v1.6.3

The following session demonstrates the gain in parallelizing the product of a random 8000×8000 matrix over $\mathbb{Z}/65521\mathbb{Z}$ with itself using 16 cores:

```
pernet@dahu34:~/soft/sage$ ./sage
SageMath version 8.9.beta8, Release Date: 2019-08-25
sage: a=random_matrix(GF(65521),8000)
sage: Parallelism()
Number of processes for parallelization:
 - linbox computations: 1
 - tensor computations: 1
sage: time b=a*a
CPU times: user 17.5 s, sys: 1.04 s, total: 18.5 s
Wall time: 18.5 s
sage: Parallelism().set("linbox",16)
sage: Parallelism()
Number of processes for parallelization:
 - linbox computations: 16
 - tensor computations: 1
sage: time b=a*a
CPU times: user 28.9 s, sys: 4.85 s, total: 33.8 s
Wall time: 2.41 s
```

Figure 1 shows computation time of three high level sage routines `b=a*a`, `a.determinant()` and `a.echelon_form()` on a large square matrix of order 20000, with varying number of cores. The speed-up relative to a single threaded run of these timings is reported in Figure 2

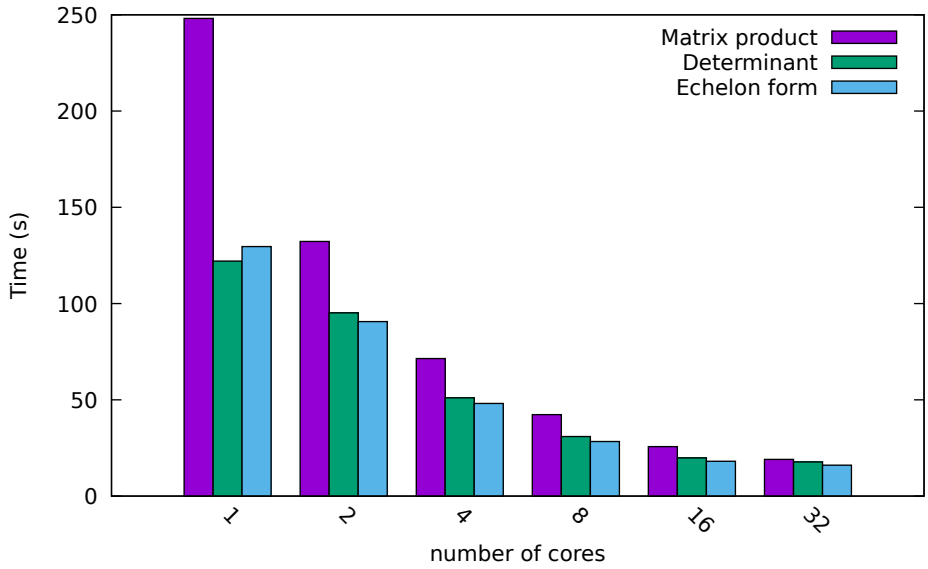

FIGURE 1. Parallel computation time for some finite field linear algebra operations in SageMath on a 32 core Intel Xeon 6130 Gold. Matrices are $20\,000 \times 20\,000$ with full rank over $\mathbb{Z}/1\,048\,573\mathbb{Z}$.

The scalability shown on Figure 2 is good but not close to the best theoretical linear speedup. This comes from the following reasons:

- first, the interface between the system SAGEMATH (running an interactive ipython) and the compiled code of the library has adds a constant overhead despite our efforts to

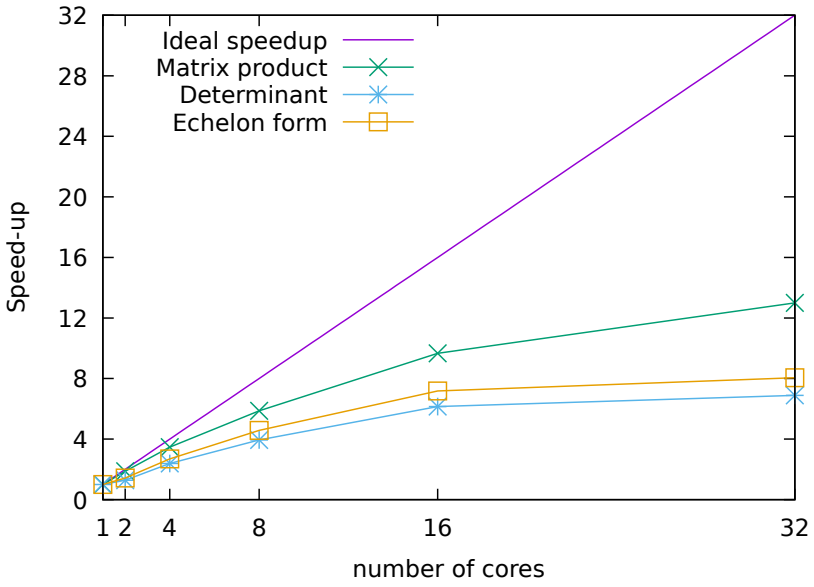

FIGURE 2. Parallel speedup for some finite field linear algebra operations in SageMath on a 32 core Intel Xeon 6130 Gold. Matrices are $20\,000 \times 20\,000$ with full rank over $\mathbb{Z}/1\,048\,573\mathbb{Z}$.

reduce it as much as possible. According to Amdhal's law, such an overhead severely penalizes the speedup measured;

- the use of sub-cubic arithmetic for the sequential matrix product tasks implies that the workload increases with the number of cores. Therefore, the true ideal speed-up curve should lie slightly below the main diagonal;
- we did not disable the turbo-boost on this server. Consequently, runs on few cores are likely executed at a higher clock frequency, hence penalizing the speedup for large number of cores.

Still, these timings show that very high performances can be attained directly from the high level interface of SAGEMATH, for instance the computation speed on 32 cores reaches 838Gfops[4] for matrix multiplication and 301 Gfops for the determinant.

2. CHALLENGES OF COMPOSING PARALLEL SOFTWARE SYSTEMS

We now report on our investigations toward exposing in SAGEMATH the parallel features of the subsystem SINGULAR, GAP, and PARI. We discuss partial successes, unique challenges, and pathways to integration in the production release of SAGEMATH. In short:

- It is to be expected that SINGULAR's parallel features will be available in the production release of SAGE sometime next spring.
- Fore the time being and in the foreseeable future it won't be possible to expose HPC-GAP parallel features to SAGE via its main library-level interface to GAP. On the other hand, they should be accessible to advanced users for hand-launched computations, with a separate installation of HPC-GAP and SAGE's legacy text-based interface to GAP, or use remote procedure calls to an HPC-GAP server which runs using the SCSCP GAP package.

[4] 1Gfops $= 10^9$ field operations per second.

- It is possible, with a custom build, to use in SAGE the development release of PARI compiled to use multithreading; some simple benchmarks demonstrate that SAGE benefits properly from the parallel features. This remains fragile however when SAGE itself uses multithreading and, as of now, one can't configure at runtime whether to use sequential or parallel computation. These issues will need to be resolved before integration in the production version of SAGEMATH.

2.1. Using a multi-threaded SINGULAR

In D5.13 we report on parallelisation of multivariate polynomial arithmetic in SINGULAR. This is achieved by outsourcing multivariate polynomial arithmetic to the C FLINT library and implementing parallelism there using of a data representation optimised for that purpose (SINGULAR's data representation of multivariate polynomials is optimised for Gröbner basis computations, which can also be run in parallel). As of now, this new implementation is available in the latest production release of FLINT, but not yet in SINGULAR's own production release, due next winter un SINGULAR's approximate yearly schedule.

There are two natural pathways for SAGE to benefit from this new work. The first approach is to bypass SINGULAR and offload multivariate arithmetic directly to FLINT. This would require significant efforts, well beyond the scope of this deliverable.

The alternative is to update SAGE to use the latest version of SINGULAR. This step is normally taken care of by the community after each production release SINGULAR and requires non trivial efforts. Indeed, SINGULAR is a standalone computer algebra system in its own right rather than an isolated library providing one or a few functions. It therefore has a large API which, even with relatively minor evolutions, requires regular updates across the whole SAGE system.

According to the approximately yearly SINGULAR release schedule, it is to be expected that the next release should occur next winter. Thanks to independent work, this new release will also include fast rational functions, factorisation, and additional polynomial orderings that directly benefit from the new parallel multivariate arithmetic. It can thus be expected that SAGE users will benefit from all these new features a couple months after.

Testing the integration in time for this deliverable would have required to update SAGE to the current development version of SINGULAR; a significant effort which was likely to have to be redone later due to still evolving API. Given that we did not foresee difficulties ♠**TO DO**: *include this statement if correct– SAGE already expose parallelism elsewhere in* SINGULAR–♠, we decided that it would be a waste of efforts at this stage.

2.2. Using HPC GAP

A similar situation applies with the progresses on HPC-GAP: for the time being, the HPC capabilities of GAP are only available through HPC-GAP as a standalone system. Exposing them through the `libgap` library would require major efforts. For details, see D5.15.

2.3. Using a multi-threaded PARI

As reported in D5.16, the number theory library PARI supports multi-threading for various operations. SAGE uses PARI for its number theory functionality. Since SAGE simply interfaces PARI, it should in theory be trivial to use PARI-MT (the multi-threading-enabled configuration of PARI) in SAGE: just enable multi-threading and that's it.

Unfortunately, some problems arise: it turns out that PARI-MT is not compatible with threads that are created outside of PARI's control. Hence, an application that is already multi-threaded cannot use PARI-MT safely without additional effort. This is in particular a problem for the documentation builder of SAGE: on the one hand, it uses PARI to create images for the documentation; on the other hand – to handle the hundreds of pages of the documentation – it uses Python's multiprocessing module, which (perhaps confusingly) also uses multiple threads.

We expect that this problem can be solved with some additional effort in either the SAGE–PARI interface or within PARI itself, typically by enabling run-time configuration of whether to use sequential or parallel computation in PARI. Nevertheless, it is easily possible to use PARI-MT in SAGE if one is careful not to build the documentation, nor to use any other kind of multi-threading. ♠**TO DO**: *some simple benchmark!*♠

REFERENCES

[1] BLUMOFE, R. D., JOERG, C. F., KUSZMAUL, B. C., LEISERSON, C. E., RANDALL, K. H., AND ZHOU, Y. Cilk: An efficient multithreaded runtime system. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 1995), PPOPP '95, ACM, pp. 207–216.

[2] GAUTIER, T., LIMA, J. V. F., MAILLARD, N., AND RAFFIN, B. Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing* (Washington, DC, USA, 2013), IPDPS '13, IEEE Computer Society, pp. 1299–1308.

[3] GAUTIER, T., ROCH, J.-L., SULTAN, Z., AND VIALLA, B. Parallel algebraic linear algebra dedicated interface. In *Proceedings of the 2015 International Workshop on Parallel Symbolic Computation* (New York, NY, USA, 2015), PASCO '15, ACM, pp. 34–43.

[4] INTEL CORPORATION. Intel threading building blocks, 2008. https://www.threadingbuildingblocks.org/.

[5] OPENMP ARCHITECTURE REVIEW BOARD. OpenMP application program interface version 5, 2018.