Efficient linear algebra on GPUs for Gröbner bases computations

Ph.D. subject in Computer Algebra and High-Performance Computing

Équipes PEQUAN and PolSys, LIP6, Sorbonne Université, 4 place Jussieu, 75005 Paris, France

Ph.D. Advisor: Stef Graillat[1],

Co-advisors: Jérémy Berthomieu[2], Théo Mary[3].

Context, scientific positioning

Modeling problems from biology, coding theory, combinatorics, robotics or aerospace engineering relies on fundamental problems such as solving polynomial systems $f_1 = \cdots = f_m = 0$ in variables $x_1, \ldots, x_n$ exactly over a finite field or rational numbers.

Solving polynomial systems is NP-hard even if the base field is finite [23, Appendix A7.2]. Thus, the end-user may ask several questions on the solution set: is it finite over the algebraic closure of the base field? In the rational case, are the complex or real solutions clustered? What is the dimension of the solution set if it is not finite?

Solving such a system comes down to computing a representation of its solution set. Furthermore, to bypass the intrinsic numerical issues generated by the non-linearity of the problem, we use mainly exact arithmetic to compute over rational numbers or in finite fields. Moreover, in the exact computation methods domain, which are thus the ones providing the highest guarantees on the quality of the result, Gröbner bases computations is the main tool to do so. Current Gröbner bases algorithms highly rely on linear algebra with large matrices (several millions of rows and columns) with a very particular structure. When this solution set has finite size D, we aim to compute a *lexicographic Gröbner basis* of the ideal spanned by $f_1, \ldots, f_m$. In the fashion of Gaussian elimination it returns a triangular system, which generically is of the form $g_n(x_n) = 0, x_{n-1} = g_{n-1}(x_n), \ldots, x_1 = g_1(x_n)$, $\deg g_n = \mathrm{D}$. Gröbner bases are a powerful tool but their computation is in general exponential in n, though some Gröbner bases are easier to compute than others. Indeed, the traditional strategy to obtain a lexicographic Gröbner basis is in two steps. First, compute a Gröbner basis of the ideal, for a *total degree* ordering using Buchberger's [13] or Faugère's F_4 [16] and F_5 [17] algorithms. Then, apply a change of ordering algorithm on it using the FGLM algorithm [19] or its faster variant, the Sparse-FGLM algorithm [20, 21]. In the generic case, this step yields $g_n, \ldots, g_1$ in essentially $\mathrm{O}(\mathrm{D}^3)$ operations. This latter algorithm guesses recurrence relations of a sequence through dedicated algorithms [2, 3, 6–9, 12, 25]. Let us notice that all these algorithms rely heavily on linear algebra routines on structured matrices. Still, their complexities are not satisfactory, mostly because the structure of the matrices does not seem to be sufficiently exploited.

Many computer algebra systems and libraries provide implementations of these polynomial system solving algorithms. We can cite FGb [18], Magma [11], Maple [29], msolve [4, 5], Singular [15] and tinygb [31]. Among these ones, only msolve, Singular and tinygb are *open-source*.

Modular arithmetic. Historically, exact arithmetic uses modular arithmetic with integer types. With the rise of Central Processing Units (CPUs) vector extensions and their performances over floating-pointer numbers compared to integers, a new line of research was developed in order to take advantage of these instructions. The idea is to perform modular arithmetic over a 26-bit prime number with double-precision floating-point number [24, 26]. Let us notice that these missing 5 bits matter. Indeed, to reach the same precision over rationals with 26-bit primes as with 31-bit primes, we need to perform 20% more computations, see [30]. Thus, this can annihilate the gain from using floating-point arithmetic and we shall pay a close attention to this.

Graphics Processing Units (GPUs) are, by design, well-suited to process large blocks of data in parallel and thus to perform linear algebra routines, more so than CPUs. Furthermore, they natively handle double-precision floating-point number arithmetic but only simulate long integer ones through short integer arith-

[1]Sorbonne Université, stef.graillat@sorbonne-universite.fr, https://www-pequan.lip6.fr/~graillat
[2]Sorbonne Université, jeremy.berthomieu@lip6.fr, https://www-polsys.lip6.fr/~berthomieu, Habilitation defense planned soon
[3]CNRS, theo.mary@lip6.fr, https://www-pequan.lip6.fr/~tmary

metic, which comes with an overhead. For instance, Nvidia CUDA and Tensor cores, natively, only have 8-bit integer types, whereas they, natively, have 64-bit floating-point types, see https://www.nvidia.com/en-us/data-center/tensor-cores/. Furthermore, it has been shown that GPUs are very efficient for correctly rounding functions evaluations [22], reinforcing even more the advantage of using floating-point arithmetic in order to simulate a modular one. Thus, we aim to take advantage of their computational power to transpose the linear algebra code of our polynomial system solver msolve to efficiently work with GPUs.

Faugère's F_4 algorithm. The F_4 algorithm builds Macaulay matrices, a special kind of structured matrices and sparse matrices. Their rows represent generators, or multiples thereof, of the ideals and their columns monomials. Thus, these rows are not completely dense. Furthermore, since several rows are given as multiples of a common polynomial, they share all their coefficients. Yet, these coefficients are dispatched pretty differently. This makes a dense representation far from optimal but a sparse one, a priori interesting.

The computation of a row echelon form of these matrices corresponds to polynomial reductions and the rows starting with many zeroes correspond to small polynomials in the ideal. Therefore, this reduction can be only be performed by following the monomial order given as an input of the algorithm. Otherwise, the polynomial interpretation of the computations can be erroneous. This row echelon form is computed by blocks and two types of blocks appear in these computations. Upper triangular ones and denses ones. On the one hand, the triangular blocks are the most convenient ones as they are easy to invert. On the other hand, the dense ones are the bottleneck of this approach.

Faugère and Mou's Sparse-FGLM algorithm. Another central theme of this Ph.D. thesis is the design of matrix–vector and matrix–matrix computations for the Sparse-FGLM algorithm.

This algorithm relies on the Wiedemann one for computing the minimal polynomial of a particular matrix M of size D, the number of solutions of the system. To do so, we compute a vector sequence $(v_i)_{0 \leqslant i < 2D}$ with v_0 random and $v_{i+1} = Mv_i$. Among the D rows of the matrix M, some are taken from the identity matrix: their only nonzero coefficient is a 1. The other ones are, a priori, dense but their coefficients can be read on the Gröbner basis output by the F_4 algorithm. An hybrid representation of this matrix is thus, in general, used. If we denote by t this number of dense rows, then the computation of $v_0, \ldots, v_{2D-1}$ require $O(tD^2)$ operations and this is the bottleneck of the Sparse-FGLM algorithm.

Ph.D. Objectives

The main objective of this Ph.D. thesis is the design of fast Gröbner bases computation algorithms, based on high performance linear algebra algorithms, in particular exploiting GPUs, and their integration into msolve in order to tackle applications challenges such as multivariate cryptography or robotics. This global goal will be decomposed into three increasingly ambitious objectives, each of which we envision taking about one year of the Ph.D.

Year 1: efficicent modular arithmetic. A first crucial goal of this Ph.D. thesis is revisiting modular arithmetic at the core of exact algorithms. The current msolve implementation on a CPU uses native integer types. However, the GPU architecture only simulates the 64-bit type through integer instructions on smaller sizes. Moreover, the emergence of increasingly fast low precision arithmetics, both with floating-point (`fp16`, `bfloat16`, `fp8`, ...) and integer (`int8`, `int4`) formats, provides new opportunities. In particular, recent Nvidia GPUs provide so-called Tensor core instructions that operate on low precision (16-bit or less) inputs but perform all internal computations in higher precision (32-bit).

The first objective will therefore be to assess which of the native types of a range of CPU and GPU hardware are the most suited for our computations depending both on the sizes of the prime integers and the target rationals after the reconstruction process. Indeed, if after performing computations modulo a certain number of primes, the rational reconstruction does not stabilize, it might be beneficial to start computing with larger primes that do not fit onto double-precision floating-point number. One approach that will be explored in particular is to represent high precision numbers as the unevaluated sum of low precision ones, in order to exploit low precision units such as GPU Tensor cores.

Year 2: high performance linear algebra for Gröbner bases. Once the basic kernels in modular arithmetic are efficient, we will turn to their use within the Gröbner bases computation itself. The objective will be to design Gröbner bases algorithms that are both efficient for handling our matrices and dedicated to GPUs. A first naive implementation of matrix–vector product on a GPU made the Sparse-FGLM step 24 times faster than our efficient CPU implementation. While this is very promising, our objective is to design an even more efficient algorithm, aiming for a factor at least 100. To do so, we want to develop more efficient matrix–matrix and matrix–thin matrix products on the GPU in order to have a block-Wiedemann approach [14, 25] of the Sparse-FGLM algorithm. Since the matrix at hand is very particular, it can be seen as the concatenation of a permutation matrix and a dense matrix after reordering the columns, we will study how to balance more efficiently the CPU load and the GPU load to iterate the product of this matrix with some vectors or very thin matrices.

This will require carefully optimizing the performance of the Sparse-FGLM and F_4 algorithms, on CPU and especially on GPU. Maximizing the granularity of the computations (e.g., the size of the matrix–vector products) and the data locality (to minimize cache misses and expensive data movements) is indeed crucial to efficiently exploit GPUs [10, 28]. In particular, we will consider Keller-Gehrig's algorithm [27], that computes these vectors in increasing buckets by densifying the matrix M and using matrix–matrix products. Thus, another objective is the design of such a GPU matrix–matrix product for this kind of matrices. Here, this product is even a square computation, thus we will study how to perform it efficiently computation- and memory-wise. The densification of the matrix allows us to compute more vectors but requires more memory at the same time. Thus, this may make the graphic card memory the roadblock of this approach. Hence, we shall keep in perspective the cost-effectiveness of the computation of the vectors through the Keller-Gehrig algorithm. We will start by focusing on the two extreme cases: when t/D is close to 0, so that the first powers of M are still relatively sparse and when $t = O(D)$ so that M is so dense that its powers cannot be much denser.

We also target the design of a sparse matrix arithmetic which is both efficient for handling our Macaulay matrices for the F_4 algorithm and dedicated to a GPU or a CPU +GPU architecture. As we plan to deal with Macaulay matrices by splitting them into blocks, we want to take advantage of the two kinds of processors to perform the computations in parallel by sending some of these blocks to the GPU.

Year 3: Gröbner bases at scale. In the final year of the Ph.D, we will aim to bring the efficient methods developed during the first two years at scale, in order to tackle very large problems whose solving will unlock new advances in critical applications.

In order to do so, we will need to handle such large matrices that even storing them on the GPU is not possible. We will therefore devise new algorithms that exploit a memory representation of these Macaulay matrices that suits our computations but also the limited RAM, a few tens of gigabytes, of a graphic card.

Moreover, computing Gröbner bases at scale will require the use of multiple GPUs and CPUs in parallel. We will therefore work on making the algorithms scalable in a parallel context. Notably, we will minimize memory communication between the CPU and the GPU, such as by adapt cache obivious storing and algorithms [1] to a larger scale such as the RAM of the graphic card.

References

[1] M. Bader and C. Zenger, "Cache oblivious matrix multiplication using an element ordering based on a Peano curve", in *Linear Algebra and its Applications* 417.2 (2006), Special Issue in honor of Friedrich Ludwig Bauer, pp. 301–313, DOI: 10/fpwdct.

[2] J. Berthomieu, B. Boyer, and J.-C. Faugère, "Linear Algebra for Computing Gröbner Bases of Linear Recursive Multidimensional Sequences", in *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '15, Bath, United Kingdom: ACM, 2015, pp. 61–68, DOI: 10/gdnj.

[3] J. Berthomieu, B. Boyer, and J.-C. Faugère, "Linear Algebra for Computing Gröbner Bases of Linear Recursive Multidimensional Sequences", in *Journal of Symbolic Computation* 83.Supplement C (2017), Special issue on the conference ISSAC 2015: Symbolic computation and computer algebra, pp. 36–67, DOI: 10/gkx4.

[4] J. Berthomieu, C. Eder, and M. Safey El Din, "Msolve: A Library for Solving Polynomial Systems", in *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation*, ISSAC '21, Virtual Event, Russian Federation: Association for Computing Machinery, 2021, pp. 51–58, DOI: 10/gk8549.

[5] J. Berthomieu, C. Eder, and M. Safey El Din, *msolve: A Library for Solving Polynomial Systems*, https://msolve.lip6.fr/, 2021.

[6] J. Berthomieu and J.-C. Faugère, "A Polynomial-Division-Based Algorithm for Computing Linear Recurrence Relations", in *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, New York, NY, USA: ACM, 2018, pp. 79–86, DOI: 10/gkx5.

[7] J. Berthomieu and J.-C. Faugère, "Polynomial-division-based algorithms for computing linear recurrence relations", in *Journal of Symbolic Computation* 109 (2022), pp. 1–30, DOI: 10/gs6r.

[8] J. Berthomieu, V. Neiger, and M. Safey El Din, "Faster change of order algorithm for Gröbner bases under shape and stability assumptions", preprint, https://hal.archives-ouvertes.fr/hal-03580736, 2022.

[9] J. Berthomieu and M. Safey El Din, "Guessing Gröbner bases of structured ideals of relations of sequences", in *Journal of Symbolic Computation* 111 (2022), pp. 1–26, DOI: 10/hpw9.

[10] P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh, "Mixed Precision Block Fused Multiply-Add: Error Analysis and Application to GPU Tensor Cores", in *SIAM Journal on Scientific Computing* 42.3 (2020), pp. C124–C141, DOI: 10/gkx7.

[11] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system. I. The user language", in *J. Symbolic Comput.* 24.3-4 (1997), Computational algebra and number theory (London, 1993), pp. 235–265, DOI: 10/ckdngx.

[12] R. P. Brent, F. G. Gustavson, and D. Y. Yun, "Fast solution of Toeplitz systems of equations and computation of Padé approximants", in *Journal of Algorithms* 1.3 (1980), pp. 259–295, DOI: 10/dwh8jx.

[13] B. Buchberger, "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms", in *SIGSAM Bull.* 10.3 (1976), pp. 19–29, DOI: 10/d2cskd.

[14] D. Coppersmith, "Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm", in *Math. Comp.* 62.205 (1994), pp. 333–350, DOI: 10/b724r7.

[15] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, *Singular 4-1-3 — A computer algebra system for polynomial computations*, 2020.

[16] J.-C. Faugère, "A New Efficient Algorithm for Computing Gröbner bases (F4)", in *Journal of Pure and Applied Algebra* 139.1 (1999), pp. 61–88, DOI: 10/bpq5dx.

[17] J.-C. Faugère, "A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5)", in *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, Lille, France: ACM, 2002, pp. 75–83, DOI: 10/bd4nnq.

[18] J.-C. Faugère, "FGb: A Library for Computing Gröbner Bases", in *Mathematical Software – ICMS 2010*, ed. by K. Fukuda, J. v. d. Hoeven, M. Joswig, and N. Takayama, Berlin, Heidelberg: Springer, 2010, pp. 84–87, DOI: 10/ckcqx9.

[19] J.-C. Faugère, P. Gianni, D. Lazard, and T. Mora, “Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering”, in *J. Symbolic Comput.* 16.4 (1993), pp. 329–344, DOI: 10/dv8t4j.

[20] J.-C. Faugère and C. Mou, “Fast Algorithm for Change of Ordering of Zero-dimensional Gröbner Bases with Sparse Multiplication Matrices”, in *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, ISSAC ’11, San Jose, California, USA: ACM, 2011, pp. 115–122, DOI: 10/fhhs56.

[21] J.-C. Faugère and C. Mou, “Sparse FGLM algorithms”, in *Journal of Symbolic Computation* 80.3 (2017), pp. 538–569, DOI: 10/gfz47c.

[22] P. Fortin, M. Gouicem, and S. Graillat, “GPU-Accelerated Generation of Correctly Rounded Elementary Functions”, in *ACM Trans. Math. Softw.* 43.3 (2016), DOI: 10/g3tk.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, USA: W. H. Freeman & Co., 1990.

[24] J. van der Hoeven, G. Lecerf, and G. Quintin, “Modular SIMD Arithmetic in Mathemagix”, in *ACM Trans. Math. Softw.* 43.1 (2016), DOI: 10/f82vvw.

[25] S. G. Hyun, V. Neiger, H. Rahkooy, and É. Schost, “Block-Krylov techniques in the context of sparse-FGLM algorithms”, in *Journal of Symbolic Computation* 98 (2020), Special Issue on Symb. and Alg. Comp.: ISSAC 2017, pp. 163–191, DOI: 10/gkx9.

[26] J. Jean and S. Graillat, “A Parallel Algorithm for Dot Product over Word-Size Finite Field Using Floating-Point Arithmetic”, in *2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2010, pp. 80–87, DOI: 10/bc9khw.

[27] W. Keller-Gehrig, “Fast algorithms for the characteristics polynomial”, in *Theoretical Computer Science* 36 (1985), pp. 309–317, DOI: 10/crh6wm.

[28] F. Lopez and T. Mary, “Mixed Precision LU Factorization on GPU Tensor Cores: Reducing Data Movement and Memory Footprint”, 2020.

[29] Maplesoft, *Maple 2019 – a division of Waterloo Maple Inc., Waterloo, Ontario*, 2019.

[30] C. Pernet, “Exact Linear Algebra Algorithmic: Theory and Practice”, in *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC ’15, Bath, United Kingdom: Association for Computing Machinery, 2015, pp. 17–18, DOI: 10/hpxb.

[31] P.-J. Spaenlehauer, *tinygb*, https://gitlab.inria.fr/pspaenle/tinygb, 2021.