Parallelization of Triangular Decompositions

Marc Moreno Maza Yuzhen Xie

August 23, 2006

Abstract

We discuss the parallelization of algorithms for solving polynomial systems symbolically. We address the following questions: How to discover geometrical information, at an early stage of the solving process, that would be favorable to parallel execution? How to ensure load balancing among the processors? We answer these questions in the context of triangular decompositions. We show that rich opportunities for parallel execution are obtained by combining modular techniques together with a solving process producing components by decreasing order of dimension.

Introduction

Since the discovery of Gröbner bases, the algorithmic advances in Commutative Algebra have made possible to tackle many classical problems in Algebraic Geometry that were previously out of reach. However, algorithmic progress is still desirable, for instance when solving symbolically a large system of algebraic non-linear equations. For such a system, in particular if its solution set consists of geometric components of different dimension (points, curves, surfaces, etc) it is necessary to combine Gröbner bases with decomposition techniques, such as triangular decompositions. Ideally, one would like each of the different components to be produced by an independent processor, or set of processors. In practice, the input polynomial system, which is hiding those components, requires some transformations in order to split the computations into sub-systems and, then, lead to the desired components. The efficiency of this approach depends on its ability to detect and exploit geometrical information during the solving process. Its implementation, which naturally must involve parallel symbolic computations, is yet another challenge.

Our work addresses two questions: How to discover geometrical information, at an early stage of the solving process, that would be favorable to parallel execution? How to ensure load balancing among the processors? We answer these questions in the context of triangular decompositions [3] which are a popular way of solving polynomial systems symbolically. They are used in geometric computations [5] such as implicitization and classification problems [8, 9]. They have interesting properties [6] for developing modular methods and stable numerical techniques.

Algorithms computing triangular decompositions tend to split the input polynomial system into subsystems and, therefore, are natural candidate for parallel implementation. However, the only such method which has been parallelized so far is the Characteristic Set Method of Wu, as reported in [1, 11]. This approach suffers from several limitations. For instance, the computation of the second component cannot start before that of the first one is completed; this is a limitation in view of coarse-grain parallelization. Actually, the operation which is parallelized in [1, 11] is the pseudo-reduction of several polynomials w.r.t. a triangular set. This is similar to most approaches in parallelizing Buchberger's algorithm [4, 2]; we view them as medium-grain parallelization.

The algorithms in [10], that we call Triade, for *TRIAngular DEcompositions*, follows a different approach, summarized in Section 1. It appears to be a natural candidate for coarse-grain parallel implementation, based on geometrical considerations.

However, several challenges remain to be considered. First, load balancing is very difficult to control due to irregular tasks. Even worse: for some input polynomial systems, resource consuming tasks may not be executed concurrently. (In characteristic zero, most examples from `www.SymbolicData.org` have a triangular decomposition consisting of a single component.) Second, data communication can be very heavy due to large intermediate results.

In order to achieve load balancing we rely on the following facts. For an input polynomial system, the Triade algorithm can generate the (intermediate or output) components by decreasing order of dimension, as explained in Section 1. As a consequence, expensive tasks (those in lower dimension) can be processed concurrently. In addition, when solving a (non-trivial) polynomial system modulo a prime integer, the number of these tasks may be sufficient for expecting a good speed-up in a parallel execution. The case of polynomial systems with integer coefficients can also benefit from these features by using the modular techniques introduced in [6].

We have developed a parallel scheme for the Triade algorithm, presented in Section 2, aiming at minimizing data communication overhead. Tasks are scheduled and updated by a *process manager*. Individual tasks are solved "lazily" by *process workers*. However, each *process worker* keeps track of enough information such that it can continue the solving of some of these tasks, when needed. We have realized a preliminary implementation on a shared memory multiprocessor. In Section 3, we report on our experimental results.

1 Intersecting Varieties with Quasi-Components

This section is an overview of the Triade algorithm [10] with an emphasis on its properties that are favorable to its parallelization. The notions of a regular chains and a quasi-component can be found in [3]. Definitions 1.1 and 1.2 introduce notions specific to the algorithm, after some notations.

Let $\mathbb{K}$ be a field and $X = x_1 < \cdots < x_n$ be ordered variables. For a subset $F \subset \mathbb{K}[X]$, we denote by $V(F)$ the zero set of F in the affine space $\overline{\mathbb{K}}^n$ where $\overline{\mathbb{K}}$ is an algebraic closure of $\mathbb{K}$. For a subset $W \subset \overline{\mathbb{K}}^n$, we denote by $\overline{W}$ the *Zariski closure* of W w.r.t. $\mathbb{K}$. For a regular chain $T \subset \mathbb{K}[X]$, we denote by $W(T)$ its quasi-component, by $\mathbf{Sat}(T)$ its saturated ideal, and, for $F \subset \mathbb{K}[X]$, we denote by $Z(F, T)$ the intersection $V(F) \cap W(T)$.

Definition 1.1. We call a *task* any couple $[F, T]$ where $F \subset \mathbb{K}[X]$ is a polynomial set and $T \subset \mathbb{K}[X]$ is a regular chain. The task $[F, T]$ is *solved* if F is empty, otherwise it is *unsolved*. By *solving* a task, we mean computing regular chains $T_1, \ldots, T_\ell$ such that we have:

$$V(F) \cap W(T) \subseteq \cup_{i=1}^{e} W(T_i) \subseteq V(F) \cap \overline{W(T)}.$$

The goal of the Triade algorithm is to solve tasks in the above sense. Moreover, a motivation in the algorithm design is to generate all regular chains (final or intermediate) by increasing order of their cardinality, and, thus by decreasing order of the dimension of their saturated ideals. However, the "basic routine" is the computation of intersections of the form $Z(\{p\}, T)$ (for a polynomial p) which may consist of components of different dimensions. Resolving this conflict of interest is achieved by a form of lazy evaluation, formalized below.

Definition 1.2. The tasks $[F_1, T_1], \ldots, [F_d, T_d]$ form a *delayed split* of the task $[F, T]$ and we write $[F, T] \longmapsto_D [F_1, T_1], \ldots, [F_d, T_d]$ if the following five properties together hold: (i) $Z(F_i, T_i) \prec Z(F, T)$, (ii) $Z(F, T) \subseteq Z(F_1, T_1) \cup \cdots \cup Z(F_d, T_d)$, (iii) $\mathbf{Sat}(T) \subseteq \mathbf{Sat}(T_i)$, (iv) $F_i \neq \emptyset \implies F \subseteq F_i$, (v) $F_i = \emptyset \implies W(T_i) \subseteq V(F)$.

Property (i) means that each "output" task $[F_i, T_i]$ is *more solved* than the "input" one $[F, T]$ (in a sense that we do not precise here and which is based on Ritt-Wu ordering for characteristic sets [10]). Properties (ii) to (v) imply:

$$V(F) \cap W(T) \subseteq \cup_{i=1}^{d} Z(F_i, T_i) \subseteq V(F) \cap \overline{W(T)}.$$

The properties of delayed splits show that solving the task $[F, T]$ reduces to solving tasks of the form $[p, T]$ where $T \cup \{t\}$ is a triangular set, but not necessarily a regular chain, or of the form $[\{p\}, T \cup \{t\}]$ where p, t are non-constant polynomials with the same main variable v and such that both $T \cup \{t\}$ and $T \cup \{t\}$ are regular chains. By means of polynomial GCDs and resultants, in some general sense defined in [10], one can design an operation $\mathsf{extend}(p, T)$ producing a delayed split of the above task $[p, T]$ and, an operation $\mathsf{decompose}(p, T \cup t)$ producing a delayed split of the above task $[\{p\}, T \cup \{t\}]$. Each of these operations satisfies the following key property: for every output task $[F_i, T_i]$ we have $F_i = \emptyset \iff |T_i| = |T| + 1$. Hence, these operations solve "lazily" in each branch where the dimension drops and solve "completely" in the others.

We sketch now a procedure solving an input task $[F, T]$ and generating all computed regular chains (final or intermediate) by decreasing order of the dimension of their saturated ideals.

This procedure uses a list U consisting of all unsolved tasks, a list S consisting of solved tasks, and an integer H which is the current size of the regular chains being computed. Initially $U = [[F, T]]$, S is the empty list and $H = |T|$. Our procedure can be sketched as follows

(1) Let V be the list of all tasks $[F', T']$ in U with $|T'| = H$.

(2) Let V' be the list of all tasks $[F', T']$ in V to which the operation $\mathsf{decompose}$ applies.

(3) If $V' \neq \emptyset$, then apply $\mathsf{decompose}$ to its elements, update the lists U and S, and go to **(1)**.

(4) If $V' = \emptyset$, then apply extend to each element in V, update the lists U and S, replace H by $H + 1$ and go to **(1)**.

All steps **(1)** to **(4)** lead naturally to parallel execution. This procedure is also different from Algorithm 2 in [10], which was meant to be executed sequentially.

Generating regular chains by decreasing size has at least two benefits. First, it allows to detect redundant components (by means of an inclusion test) at an early stage of the solving process. Indeed, if $W(T_1)$ contains $W(T_2)$ then we must have $|T_1| \leq |T_2|$. Second, it forces the algorithm to delay the computations in lower dimension toward the end of the solving process, which increases the opportunities for parallelization. Indeed, when computing $\mathsf{decompose}(p, T \cup t)$ the larger is $|T|$, the more expensive are calculations modulo $\mathbf{Sat}(T)$. See for instance the complexity results in [7].

2 Parallel Implementation Scheme

Based on the procedure of Section 1, we propose now an implementation scheme for parallel triangular decompositions. We denote by P_0 the *Process Manager*. Any other process worker will be denoted by some P_i for $i > 0$. Each task $[F_i, T_i]$ has a unique *task identifier (TID)*, and each worker P_i owns a unique *worker identifier (WID)*. The manager P_0 maintains a `taskTable`, which contains the tasks that are already solved, and those which are waiting (to be selected). For each task, we also record its *TID* and *WID* (the worker that has produced it). Every process worker P_i records all the tasks that P_i has computed in a `localTaskTable` (P_i stores their actual values and not only their identifiers). Recall that the key idea of the algorithm is to *solve by decreasing order of dimension*. To this effect, we have a global variable (visible by all workers) H taking successively the values $|T|, |T|+1, \ldots$. When $H = k$, then all the regular chains in the triangular decomposition of $V(F)$ which have less than k elements have already been computed. In addition, the algorithm is currently computing those regular chains which have k elements. We describe below the algorithm of P_0 with the input task $[F, T]$. Initially $H := |T|$.

(1) Select all the tasks which can lead to components of dimension $n - H$, denoted as $R_1, \ldots, R_s$.

(2) Find s workers to solve these tasks. This may result in launching new workers, but try to solve (as much as possible) each task on the worker that created it (when this is possible, only the *TID* needs to be sent to the worker).

(3) Collect all the tasks (solved or unsolved) produced by the workers and update `taskTable`.

(4) If all the tasks are solved, send a `stop` message to each worker and halt, otherwise $H := H + 1$ and go to **(1)**.

We describe now the algorithm of process worker P_i. Initially `localTaskTable` $:= \{\ \}$.

(1) Receive a task, or a message from manager P_0. In case of a "stop" message, then halt.

(2) In case of a *TID*, look up the `localTaskTable` for this task, say $[F_j, T_j]$. Compute a delayed split of $[F_j, T_j]$ such that all unsolved tasks are contained in components of dimension (strictly) less than $n - H$. Update the output tasks and `localTaskTable` by removing redundant ones.

(3) Send the output tasks to manager P_0 and go to **(1)**.

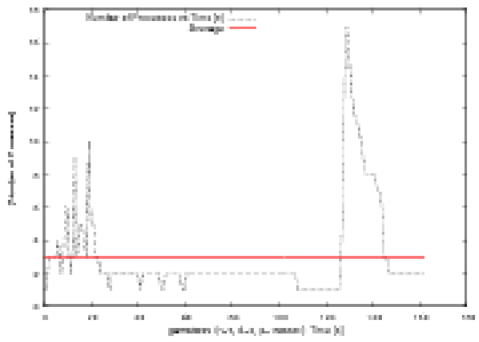
As much as possible we let process workers continue solving the tasks they have produced, so as to reduce the data communication overhead. Besides, new worker processes are activated only when the number of selected tasks exceeds the number of running workers. This strategy helps minimizing the cost of dynamic process creation and management.

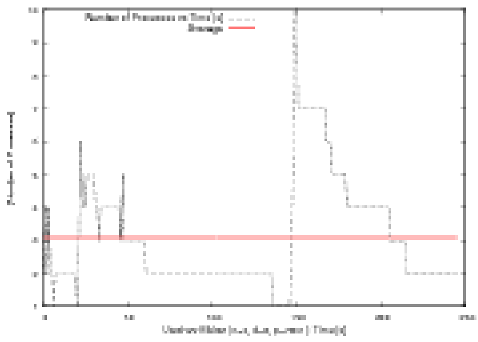
3 Experimentation

We have realized a preliminary implementation of the algorithm in Section 1 on a shared memory multiprocessor, AuthenticAMD with four CPUs (2390MHz) and 8 GB total memory. Implementation is based on the existing `BasicMath` library written in ALDOR, see `www.aldor.org`, which is a sequential implementation of the Triade algorithm. We have created two binary modules: `Process Manager` and `Process Worker`. The latter one is activated and terminated dynamically by the former one as soon as a delayed split computation is required. Between theses binaries, data communication is done through shared memory segments, which are efficient ways for System V inter process communication. To this end, tasks, and thus multivariate polynomials, are tuned into machine integer arrays through Kronecker substitution.

The polynomial systems that are used in this experimentation are taken from `www.SymbolicData-.org`. For each system, n denotes the number of variables, d is the total degree of the polynomial system and p is the prime number used in both the sequential and the parallel solving. We report in the following figures two examples, namely: gametwo5 and Uteshev-Bikker. For each of them: (1) we plot the number of processes acting at a time during the whole solving procedure, and (2) we show the average number of processes. Although the speed-up ratio we gained is between 1.5 and 2 by comparing with the sequential implementation in ALDOR, this is very promising for such a coarse-grain parallel implementation, where truly expensive tasks can be processed in parallel. It is noticed that the average number of processes in the entire solving procedure is about 3.

The implementation of the scheme in Section 2 is work in progress. In our current implementation, branching one expensive task into two expensive tasks relies only on the *D5 Principle*. Applying factorization into irreducible polynomials should improve the speed-up ratio. Further more, we need to integrate fine-grain parallelization, for GCD/resultant computations, which should improve the time and space efficiency of our parallel triangular decompositions.





References

[1] I. A. Ajwa. *Parallel Algorithms and Implementations for the Gröbner Bases Algorithm and the Characteristic Set Method*. PhD thesis, Kent State University, Kent, Ohio, 1998.

[2] G. Attardi and C. Traverso. Strategy-accurate parallel Buchberger algorithms. *Journal of Symbolic Computation*, 21(4):411–425, 1996.

[3] P. Aubry, D. Lazard, and M. Moreno Maza. On the theories of triangular sets. *J. Symb. Comp.*, 28(1-2):105–124, 1999.

[4] S. Chakrabarti and K. Yelick. Distributed data structures and algorithms for Gröbner basis computation. *Lisp and Symbolic Computation: An International Journal*, 7:147–172, 1994.

[5] F. Chen and D. Wang, editors. *Geometric Computation*. Number 11 in Lecture Notes Series on Computing. World Scientific Publishing Co., Singapore, New Jersey, 2004.

[6] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, ACM Press, 2005.

[7] X. Dahan, M. Moreno Maza, É. Schost, and Y. Xie. On the complexity of the D5 principle. In *Proc. of* Transgressive Computing 2006, Granada, Spain, 2006.

[8] M.V. Foursov and M. Moreno Maza. On computer-assisted classification of coupled integrable equations. In *proceedings of ISSAC 2001*, pages 129–136, London, Ontario, 2001. ACM Press.

[9] I. A. Kogan and M. Moreno Maza. Computation of canonical forms for ternary cubics. In Teo Mora, editor, *Proc. ISSAC 2002*, pages 151–160. ACM Press, July 2002.

[10] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. Presented at the MEGA-2000 Conference, Bath, England.

[11] Y.W. Wu, G.W. Yang, H. Yang, W.M. Zheng, and D.D. Lin. A distributed computing model for wu's method. *Journal of Software (in Chinese)*, 16(3), 2005.

ORCCA, University of Western Ontario, London, Ontario, Canada