Component-level Parallelization of Triangular Decompositions

Marc Moreno Maza
ORCCA, University of Western Ontario (UWO)
London, Ontario, Canada
moreno@orcca.on.ca

Yuzhen Xie
ORCCA, University of Western Ontario (UWO)
London, Ontario, Canada
yxie@orcca.on.ca

ABSTRACT

We discuss the parallelization of algorithms for solving polynomial systems symbolically by way of triangular decompositions. We introduce a component-level parallelism for which the number of processors in use depends on the geometry of the solution set of the input system. Our long term goal is to achieve an efficient multi-level parallelism: coarse grained (component) level for tasks computing geometric objects in the solution sets, and medium/fine grained level for polynomial arithmetic such as GCD/resultant computation within each task.

Component-level parallelization of triangular decompositions belongs to the class of dynamic irregular parallel applications, which leads us to address the following question: How to exploit geometrical information at an early stage of the solving process that would be favorable to parallelization? We report on the effectiveness of the approaches that we have applied, including "modular methods", "solving by decreasing order of dimension", "task pool with dimension and rank guided scheduling". We have extended the AldorProgramming language to support multiprocessed parallelism on SMPs and realized a preliminary implementation. Our experimentation shows promising speedups for some well-known problems and proves that our component-level parallelization is practically efficient. We expect that this speedup would add a multiplicative factor to the speedup of medium/fine grained level parallelization as parallel GCD and resultant computations.

Categories and Subject Descriptors: F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity. F.2.2 [Nonnumerical Algorithms and Problems]: Computations on discrete structures; Sequencing and scheduling.

General Terms: Algorithms, Theory, Experimentation, Performance.

PASCO'07, July 27-28, 2007, London, Ontario, Canada.

Keywords: Polynomial system solving, Triangular decompositions, Parallelization, Component-level.

1. INTRODUCTION

Symbolic solvers of polynomial systems are powerful tools in scientific computing: they are well suited for problems where the desired output must be exact and they have been applied successfully in areas like digital signal processing, robotics, theoretical physics, cryptology with many important outcomes. See [15] for an overview of these applications. The implementation of symbolic methods is, however, a highly difficult task. Indeed, they are extremely time consuming when applied to large examples. Moreover, intermediate expressions can grow to enormous size and may halt the computations, even if the result is of moderate size.

The increasing availability of parallel computer architectures, from SMPs to multi-core laptops, has revitalized the need for developing mathematical algorithms and software capable of exploiting these new computing resources. This need is even more dramatic in the case of symbolic computations which offer exciting, but highly complex challenges to computer scientists. This paper aims at investigating new directions in the parallelization of symbolic solvers for polynomial systems.

Ideally, one would like that each component of the solution set of a polynomial system could be produced by an independent processor, or a set of processors. In practice, the input polynomial system, which is hiding those components, requires some transformations in order to split the computations into subsystems 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 involves parallel symbolic computations, is yet another challenge.

Several symbolic algorithms provide a decomposition of the solution set of any system of algebraic equations into components (which may be irreducible or with weaker properties): primary decompositions [14, 32], comprehensive Gröbner bases [35], triangular decompositions [36, 20, 21, 29, 34] and others. These algorithms tend to split the input polynomial system into subsystems and, therefore, seem to be natural candidates for a *component-level* parallelization.

Unfortunately, such a parallelization is very likely to be unsuccessful, bringing no practical speedup w.r.t. comparable sequential implementations of the same algorithms. Indeed, even if computations split into sub-problems which can be processed concurrently, the computing cost of the

corresponding tasks are extremely irregular. Even worse: for input polynomial systems with coefficients in the field $\mathbb{Q}$ of rational numbers, a single heavy task may dominate the whole solving process, leading essentially to no opportunities for component-level parallel execution. This phenomenon follows from the following observation. For most polynomial systems with coefficients in $\mathbb{Q}$ that arise in theory or in practice, see for instance `www.SymbolicData.org`, the solution set can be described by a single component! The theoretical justification is given by the celebrated *Shape Lemma* [5] for systems with finitely many solutions.

We show, in this paper, how to achieve a successful *component-level* parallelization for polynomial systems including for the case of rational number coefficients. Among the algorithms that decompose the solution set of a polynomial system into components, we consider one computing triangular decompositions, called Triade [29]. The first reason for this choice is that, triangular decompositions of polynomial systems with coefficients in $\mathbb{Q}$ can be reduced to triangular decompositions of polynomial systems modulo a prime number [11], bringing rich opportunities for parallel execution. We discuss in Section 2 the main features of this algorithm that are relevant to parallelism. The second reason is that this algorithm has been implemented in the ALDOR language [2] and in the computer algebra systems AXIOM [19] and MAPLE [27] as the `RegularChains` library [23]. This provides us with useful tools for our experimentation work. The third and main reason is that the Triade algorithm can generate the (intermediate or output) components by decreasing order of dimension. As we show in Section 3, this allows us to exploit the opportunities for parallel execution created by modular techniques, leading to successfully component-level parallel execution.

Our objective is to develop a parallel solver for which the number of processors in use depends on the *intrinsic complexity* of the input system, that is on the geometry of its solution set. This approach is not aimed to bring scalability. For instance, for systems over Z/pZ with finitely many solutions, if the output consists of s components with similar degrees, we cannot expect a speed-up much larger than s by relying only on a component-level parallelism. We do not aim neither at replacing the previous approaches for parallel polynomial system solving. On the contrary, we aim at adding an extra level of parallelism.

The parallelization of two other algorithms for solving polynomial systems symbolically have already been actively studied. First, Buchberger's algorithm for computing Gröbner bases, see for instance [6, 7, 8, 13, 3, 25]. Second, the Characteristic Set Method of Wu [36], see [1, 37, 38]. In all these works, the parallelized operation is polynomial reduction (or simplification). More precisely, given two polynomial sets A and B (with some conditions on A and B, depending on the algorithm) the reductions of the elements of A by those of B are executed in parallel.

The Triade algorithm also has a *polynomial simplification level* which relies on polynomial GCDs and resultants. The parallelization of such computations is reported in [31, 18]. The addition of this second level to the Triade algorithm is work in progress.

In Section 2, we present a *task model* associated with each triangular decomposition computed by the Triade algorithm. We review also how this algorithm makes use of geometrical information discovered during the computations. The techniques that are applied to create the component-level parallelism and to control the feature of tasks in favor of parallelization are introduced in Section 3. Our heuristically efficient *Task Pool with Dimension and Rank Guided scheduling* (TPDRG) is reported in Section 4. In the remaining sections, we report our preliminary implementation and experimentation. We have extended the ALDOR programing language for multi-processed parallel programing on SMPs and realized a preliminary implementation of this component-level parallel algorithm based on the `BasicMath` library [17]. We have conducted an intensive experimentation on some well-known problems. A comparison on the practical efficiency between our TPDRG scheduling and the generally good *Greedy* scheduling has also been performed. These help in evaluating the efficiency of our implementation and reveal its limitation as well. In the conclusion, we discuss the potential to extend this work to achieve efficient multi-level parallelization for triangular decompositions.

2. TASK MODEL

We discuss in this section the main features of the Triade algorithm that are relevant to parallelism. After some notations, we recall in Definition 1 the notion of a *regular chain*, which appears in most algorithm computing triangular decompositions. Then, we review the notions that are specific to the Triade algorithm such as that of a *Task*, Definition 2 and that of a *delayed split*, Definition 3. They are well-adapted to describe the relations between the intermediate computations during the solving of a polynomial system. Algorithm 1 is the top-level procedure of the Triade algorithm: it manages a *task pool*. The tasks are transformed by means of a sub-procedure (Algorithms 2 and 3) which is dedicated to "simple tasks". The execution of such a simple task can be highly irregular and dynamic. It can also generate other tasks. Therefore, Algorithms 1, 2 and 3 may not lead to successful parallel execution. In fact, we will adapt them in Section 3 for this purpose.

NOTATION 1. *Let $\mathbb{K}$ be a field and $X = x_1 < \cdots < x_n$ be n 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 $\mathbb{K}$. The polynomial p is said* regular *modulo the ideal $\langle F \rangle$ if it is neither zero, nor a zero-divisor modulo $\langle F \rangle$. For a subset $W \subset \overline{\mathbb{K}}^n$, we denote by $\overline{W}$ the* Zariski closure *of W w.r.t. $\mathbb{K}$, that is, the intersection of the $V(G)$ containing $W(T)$ for all $G \subseteq \mathbb{K}[X]$.*

Let $p \in \mathbb{K}[X]$ be a non-constant polynomial. We denote by mvar(p) the main variable *(or largest variable) of p, by init(p) the* initial *(or leading coefficient w.r.t. mvar(p)) of p and by rank(p) the* rank *of p, that is, v^d where $v = mvar(p)$ and d is the degree of p w.r.t. v. For two ranks $v_1^{d_1}$ and $v_2^{d_2}$ we write $v_1^{d_1} \prec v_2^{d_2}$ whenever $v_1 < v_2$ or, $v_1 = v_2$ and $d_1 < d_2$ hold.*

Let $T \subset \mathbb{K}[Y]$ be a triangular set*, that is, a set of non-constant polynomials with pairwise different main variables. Let h_T be the product of the initials of T. We denote by $W(T)$ the* quasi-component *of T, that is, $V(T) \setminus V(h_T)$, in other words, the set of the points in $V(T)$ which do not cancel any of the initials of T. For $F \subset \mathbb{K}[X]$, we denote by $Z(F,T)$ the intersection $V(F) \cap W(T)$. We denote by mvar(T) the set of the mvar(t) and by rank(T) the set of the rank(t), for all t in T. A variable from X is said* algebraic *w.r.t. T if it belongs to mvar(T).*

Finally, we denote by Sat(T) the saturated ideal of T, *which is defined as follows. If T is empty then Sat(T) is defined as the trivial ideal $\langle 0 \rangle$ otherwise it is the ideal of all $p \in \mathbb{K}[Y]$ such that there exists an integer e such that $h_T^e p$ belongs to $\langle T \rangle$. The ideal Sat(T) has two important properties. First, its zero set satisfies the following: $V(Sat(T)) = \overline{W(T)}$. Secondly, if Sat($T$) is a proper ideal, then it is equidimensional and its dimension is equal to $n - |T|$, see [20].*

DEFINITION 1. *The triangular set T is a* regular chain *if either T is empty or: T is not empty, $T \setminus \{T_{\max}\}$ is a regular chain, and the initial of $T_{\max}$ is regular w.r.t. Sat($T \setminus \{T_{\max}\}$), where $T_{\max}$ is the polynomial in T with maximum rank. A finite family $\mathcal{T}$ of regular chains of $\mathbb{K}[X]$ is a* triangular decomposition *of $V(F)$ if we have*

$$V(F) = \cup_{T \in \mathcal{T}} \ W(T).$$

DEFINITION 2. *We call a* task *any couple $[F, T]$ where F is a finite subset of $\mathbb{K}[X]$ 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 $T_1, \ldots, T_e$ such that we have:*

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

Most algorithms computing triangular decompositions consist of procedures that take a task $[F_0, T_0]$ as input and returns zero, one or several tasks $[F_1, T_1], \ldots, [F_e, T_e]$. Then, solving an input polynomial system F_0 is achieved by calling one of these procedures with $[F_0, \emptyset]$ as input and obtaining "solved tasks" $[\emptyset, T_1], \ldots, [\emptyset, T_e]$ as output, such that $T_1, \ldots, T_e$ solves $[F_0, \emptyset]$ in the sense of Definition 2.

Therefore, given an algorithm A for computing triangular decompositions, it is natural to associate with each input polynomial system F_0 a *task tree* $G_A(F_0)$ whose vertices are tasks such that there is an arrow from any task $[F_i, T_i]$ to any task $[F_j, T_j]$ if task $[F_j, T_j]$ is among the output tasks of a procedure called on $[F_i, T_i]$; moreover, each internal node $[F_i, T_i]$ has a weight equal to the the (estimated) running time for computing the children of $[F_i, T_i]$. The longest path (summing the weights along the path) from the root to a leaf, called *critical path of* $G_A(F_0)$ and often denoted by T_∞, represents the minimum running time for a parallel execution of $A(F_0)$ on infinitely many processors. (Here we do not consider communication costs and scheduling overheads, for simplicity.) The sum of the all weights in $G_A(F_0)$, called the *work of* $G_A(F_0)$ and often denoted by T_1, represents the minimum running time for a sequential execution of $A(F_0)$.

It is well known that most of algorithms decomposing polynomial systems into components (irreducible, equidimensional, ...) have to face the problem of *redundant components*, which may occur in the output or at intermediate stages of the solving process. This is a central question when computing triangular decompositions, see [4] for a discussion of this topic. Removing redundant components is also an important issue in other symbolic decomposition algorithms such as the one of [22] and also for numerical ones [33]. Being able to remove redundant components at an early stage of the computations helps reducing the work of $G_A(F_0)$ and, possibly its critical path. One of the motivations in the design of the Triade algorithm [29] is to handle efficiently redundant components.

For any input task $[F, T]$ the main procedure of the Triade algorithm, called Triangularize(F, T), solves $[F, T]$ in the sense of Definition 2. This procedure reduces to the situation where F consists of a single polynomial p. One could expect that such an operation, say Decompose(p, T), should return regular chains $T_1, \ldots, T_\ell$ solving the task $[\{p\}, T]$. In fact, we shall explain now why this would not meet our requirement of handling redundant components efficiently.

Observe that $W(T_1) \subseteq W(T_2)$ implies $|T_2| \leq |T_1|$. It follows that, during the solving process, all the (final or intermediate) regular chains should be generated by increasing order of their cardinality, that is, by decreasing order of the dimension of their saturated ideals, in order to remove the redundant ones as soon as possible. Returning to the specifications of the operation Decompose(p, T), observe that $V(p) \cap W(T)$ could contain components of different dimension. (This will happen when $V(p)$ contains some of the irreducible components of $\overline{W(T)}$, but not all of them.) Therefore, it is not desirable for the operation Decompose(p, T) to solve the task $[\{p\}, T]$ in one step. Instead, Decompose(p, T) should compute the quasi-components of $V(p) \cap W(T)$ of maximum dimension and postpone the computation of the other quasi-components. This is made possible by a form of lazy evaluation, formalized by Definition 3, after Notation 2.

NOTATION 2. *Let T_1, T_2 be two regular chains. We write rank(T_1) $\prec$ rank(T_2) whenever rank(T_2) is a proper subset of rank(T_1), or when $v_1^{d_1} \prec v_2^{d_2}$, where $v_1^{d_1}$ (resp. $v_2^{d_2}$) is the smallest element of rank(T_1) $\setminus$ rank(T_2) (resp. rank(T_2) $\setminus$ rank(T_2)). When neither rank(T_1) $\prec$ rank(T_2) nor rank(T_2) $\prec$ rank(T_1) hold, we write rank(T_1) $\simeq$ rank(T_2). Let F_1, F_2 be finite subsets of $\mathbb{K}[X]$. We write $[F_1, T_1] \prec [F_2, T_2]$ either if rank(T_1) $\prec$ rank(T_2) holds, or if rank(T_1) $\simeq$ rank(T_2) holds and there exists $f_1 \in F_1$ such that rank(f_1) $\prec$ rank(f_2) for all $f_2 \in F_2$. Clearly, any sequence of tasks $[F_0, T_0], \ldots$, such that $[F_i, T_i] \prec [F_{i+1}, T_{i+1}]$ holds for all i, is finite.*

DEFINITION 3. *The tasks $[F_1, T_1], \ldots, [F_e, T_e]$ form a* delayed split *of the task $[F, T]$ and we write*

$$[F, T] \longmapsto_D \ [F_1, T_1], \ldots, [F_e, T_e]$$

if for all $1 \ \leq i \leq e$ we have $[F_i, T_i] \prec [F, T]$ and the following holds

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

Below, we highlight the main features of the procedure Decompose(p, T) that are relevant to the rest of the paper. First, for a polynomial p and a regular chain T, such that p is not zero modulo Sat(T), the procedure Decompose(p, T) returns a delayed split of the task $[\{p\}, T]$. Algorithm 1 implements the procedure Triangularize by means of the procedure Decompose. Based on the specifications of Decompose, the validity of this algorithm is easy to check and is established in [29]. Note that our pseudo-code uses the syntax of the Computer Algebra System AXIOM [19]. In particular, we use indentation to denote blocks. Moreover, each of our algorithms generates a sequence of items which are returned one by one in the output flow by means of **yield** statements.

ALGORITHM 1.

Input: *a task* $[F, T]$

Output: *regular chains* $T_1, \ldots, T_e$ *solving* $[F, T]$ *in the sense of Definition 2*

```
Triangularize(F, T) == generate
1  R := [[F, T]]
2  # R is a list of tasks
3  while R ≠ [] repeat
4     choose and remove a task [F_1, U_1] from R
5     F_1 = ∅ ⟹ yield U_1
6     choose a polynomial p ∈ F_1
7     G_1 := F_1 \ {p}
8     p ≡ 0  mod Sat(U_1) ⟹ R := cons ([G_1, U_1], R)
9     for [H, T] ∈ Decompose(p, U_1) repeat
10       R := cons ([G_1 ∪ H, T], R)
```

The key notion used by the procedure Decompose is that of a polynomial GCD modulo a regular chain, see Definition 4. This notion strengthens that introduced by Kalkbrener in [20] and extends that of a polynomial GCD modulo a triangular set introduced in [30].

DEFINITION 4. *Let* p, t, g *be non-zero polynomials and* T *be a regular chain. Assume that* p *and* t *are non-constant and have the same main variable* v*. Assume that* $v \notin mvar(T)$*, that* $init(p)$ *is regular w.r.t.* $Sat(T)$ *and that* $T \cup \{t\}$ *is a regular chain. Then, the polynomial* g *is a* GCD of p and t w.r.t. T *if the following properties hold:*

(G_1) *g belongs to the ideal generated by p, t and $Sat(T)$,*

(G_2) *the leading coefficient h_g of g w.r.t. v is regular modulo $Sat(T)$,*

(G_3) *if $mvar(g) = v$ then p and t belong to $Sat(T \cup \{g\})$.*

More generally, a sequence of pairs $G = (g_1, T_1), \ldots, (g_e, T_e)$*, where* $g_1, \ldots, g_e$ *are polynomials and* $T_1, \ldots, T_e$ *are regular chains, is a* GCD sequence of p and t w.r.t. T *if the following properties hold:*

(G_4) *for all $1 \leq i \leq e$, if $|T_i| = |T|$ then g_i is a GCD of p and t modulo T_i,*

(G_5) *we have $W(T) \subseteq \cup_{i=1}^{e} W(T_i) \subseteq \overline{W(T)}$.*

Four procedures of Triade are essential to the rest of the paper. Their specifications are reviewed in Notation 3.

NOTATION 3. *Let* p, t, T *be as in Definition 4. The procedure* $\mathsf{GCD}(p, t, T)$ *computes a GCD sequence of* p and t w.r.t. T*. These GCD computations allow testing whether a polynomial f is regular modular $Sat(T)$. Given a polynomial f, the procedure RegularizeInitial(f, T) returns regular chains $T_1, \ldots, T_e$ such that for all $1 \leq i \leq e$, the polynomial f is congruent to constant modulo $Sat(T_i)$, or congruent to a non-constant polynomial f_i modulo $Sat(T_i)$, whose initial $init(f_i)$ is regular modulo $Sat(T_i)$. Given a triangular set $S \subset \mathbb{K}[X]$ the procedure* $\mathsf{Extend}(S)$ *returns regular chains $T_1, \ldots, T_e$ satisfying $W(S) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq \overline{W(S)}$. Finally, we denote by* $\mathsf{Reduce}(f, T)$ *the polynomial defined as follows: if $f \in \mathbb{K}$, then* $\mathsf{Reduce}(f, T)$ *is f; if $f \in Sat(T)$, then* $\mathsf{Reduce}(f, T)$ *is 0 otherwise it is* $\mathsf{Reduce}(h, T)\, mvar(f) + \mathsf{Reduce}(g)$*, where $h = init(f)$ and $g = f - init(f) rank(f)$.*

Algorithm 3 states the algorithm for $\mathsf{Decompose}(p, T)$. It relies on a sub-procedure given by Algorithm 2.

ALGORITHM 2.

Input: p, T, t *as in Definition 4.*

Output: *a delayed split of* $[\{p\}, T \cup \{t\}]$.

```
AlgebraicDecompose(p, T, t) == generate
1   Let h_T be the product of the initials in T
2   for [g_i, T_i] ∈ GCD(t, p, T) repeat
3      |T_i| > |T| ⟹
4         for T_{i,j} ∈ Extend(T_i ∪ {h_T t}) repeat
5            yield [p, T_{i,j}]
6      g_i ∈ K ⟹ iterate
7      mvar(g_i) < v ⟹ yield [{g_i, p}, T_i ∪ {t}]
8      deg(g_i, v) = deg(t, v) ⟹ yield [∅, T_i ∪ {t}]
9      yield [∅, T_i ∪ {g_i}]
10     yield [{init(g_i), p}, T_i ∪ {t}]
```

ALGORITHM 3.

Input: *a polynomial p and a regular chain T such that $p \notin Sat(T)$.*

Output: *a delayed split of* $[\{p\}, T]$.

```
Decompose(p, T) == generate
1   for C ∈ RegularizeInitial(p, T) repeat
2      f := Reduce(p, C)
3      f = 0 ⟹ yield [∅, C]
4      f ∈ K ⟹ iterate
5      v := mvar(f)
6      v ∉ mvar(C) ⟹
7           yield [{init(f), p}, C]
8           for D ∈ Extend(C ∪ {f})
9              repeat yield [∅, D]
10     for [F, E] ∈ AlgebraicDecompose(f, C_{<v} ∪ C_{>v}, C_v)
11        repeat yield [F, E]
```

The proof of Algorithms 2 and 3 relies fundamentally on Proposition 1. In broad words, this result states that the common zeros of p and t contained in $W(T)$ are "essentially" given by $W(T \cup \{g\})$, where g is a GCD of p and t w.r.t. T in the sense of Definition 4. See [29] for detail.

PROPOSITION 1. *Let* p, t, g, T *be as in Definition 4. If g is a GCD of p and t w.r.t. T and $mvar(g) = v$ holds, then we have*

$$[[\{p\}, T \cup \{t\}] \longmapsto_D [\emptyset, T \cup \{g\}], [\{h_g, p\}, T \cup \{t\}].$$

The following fundamental proposition is easily checked from the pseudo-code of Algorithm 2 and Algorithm 3.

PROPOSITION 2. *Let* $[F, E]$ *be any task returned by Algorithm 3. Then we have:*

(H_1) *either $|E| = |T|$ and $F = \emptyset$,*

(H_2) *or $|E| = |T|$ and F contains a polynomial which is regular w.r.t. $Sat(T)$,*

(H_3) *or $|E| > |T|$.*

COROLLARY 1. *Let $[F, E]$ be a task returned by Algorithm 3. If $\overline{W(E)}$ is a component of $V(p) \cap \overline{W(T)}$ with maximum dimension, then the task $[F, E]$ is solved, that is, $F = \emptyset$.*

Solving by decreasing order of dimension. It follows from Corollary 1 that the tasks in Algorithm 1 can be chosen such that the regular chains output by this algorithm are generated by increasing size. To do so, we assign to each task $[F, T] \in R$ in Algorithm 1 an upper bound $m([F, T])$ for the height of the regular chains solving $[F, T]$ in the sense of Definition 2. This upper bound is simply computed as follows. If a polynomial $f \in F$ has been shown to be regular w.r.t. T (See Proposition 2) then $m([F, T]) := |T| + 1$ otherwise $m([F, T]) := |T|$. Then, we say that a task $[F_1, T_1]$ has a *higher priority* than a task $[F_2, T_2]$ if either $m([F_1, T_1]) \leq m([F_2, T_2])$ holds, or $m([F_1, T_1]) = m([F_2, T_2])$ and $[F_1, T_1] \prec [F_2, T_2]$ hold. Sorting the tasks in the list R w.r.t. this ordering allows us to solve by decreasing order of dimension and therefore to handle redundant components efficiently. The performances of our inclusion test are reported in [9].

3. PARALLELIZATION

One could think of deriving a parallel scheme from Algorithm 1 by running the procedure Triangularize(F, T) on one processor and running each call to Decompose on any other available processor, following a greedy scheduling. As mentioned in the Introduction, such parallelization is very likely to be unsuccessful, bringing no practical speed-up w.r.t. comparable sequential implementations of the same algorithms. In characteristic zero, this mainly follows from the fact, for most polynomial systems, the solution set can be described by a single component, though not necessarily irreducible. In prime characteristic, however, even if a single component suffices, it is more likely that polynomials factorize and thus that components split.

Using modular techniques. The previous remark suggests the use of modular techniques for computing triangular decompositions. We rely on the algorithm proposed in [11]. For a given input square system $F \subset \mathbb{Q}[x_1, \ldots, x_n]$ this algorithm computes the simple points of $V(F)$ in four steps:

(S_1) compute a prime number p such that $V(F)$ can be reconstructed, with high probability, from $V_p := V(F \bmod p)$, the zero-set of F regarded in $Z/pZ[x_1, \ldots, x_n]$,

(S_2) compute a triangular decomposition of V_p,

(S_3) compute the equiprojectable decomposition of $_p$,

(S_4) reconstruct by Hensel lifting the equiprojectable decomposition of $V(F)$.

As reported in [11], the second step has the dominant cost. Therefore, we focus on computing triangular decompositions of polynomial systems with coefficients in a finite field.

Our test suite. Table 1 contains data about 7 well-known test systems that we use through the experiments reported in this article. All of them are polynomial systems over $\mathbb{Q}$: for each we give its number of n of equations, its total degree d, the prime number p provided by the above Step (S_1) and the list of the degrees of the triangular decomposition computed at Step (S_2) by the Triade algorithm. We stress the fact that each of these systems, except Cohn2, is equiprojectable, that is, its equiprojectable decomposition consists of a single component. Hence, for a direct computation $\mathbb{Q}$, the computations may not split. Therefore, our modular approach has created opportunities for parallel execution.

Sys	Name	n	d	p	*Degrees*
1	eco6	6	3	105761	[1,1,2,4,4,4]
2	eco7	7	3	387799	[1,1,1,1,4,2, 4,4,4,4,4,2]
3	CassouNogues2	4	6	155317	[8]
4	CassouNogues	4	8	513899	[8,8]
5	Nooburg4	4	3	7703	[18,6,6,3,3,4, 4,4,4,2,2,2, 2,2,2,2,2,1, 1,1,1,1]
6	UteshevBikker	4	3	7841	[1,1,1,1,2,30]
7	Cohn2	4	6	188261	[3,5,2,1,2,1,1, 16,12,10,8,8, 4,6,4,4,4,4,2, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1]

Table 1: Features of the polynomial systems

Regularizing initials for controlling task irregularity. A call to the procedure Decompose as given by Algorithm 3 may result in unpredictable amount of work. Indeed, since the initial of p may not be regular w.r.t. $\mathrm{Sat}(T)$, the polynomial f computed at Line 2 may have a different main variable than p. Hence we cannot predict the main variables and degrees of the input polynomials in the calls to AlgebraicDecompose and Extend. It could be the case that these calls lead to inexpensive operations, say polynomial GCDs of univariate polynomials of low degrees, whereas the regular chain contains very large polynomials in many variables. Therefore, Decompose(p, T) may lead to inexpensive computations but expensive data communication. In order to control this phenomenon, we strengthen the notion of a task in Definition 5: the **initial** of every polynomial $f \in F$ in a task $[F, T]$ must be **regular** w.r.t. $\mathrm{Sat}(T)$. The motivation of this Definition is twofold. First, we want to anticipate which operations will be performed by Algorithm 3. Second, we want to force light-load calls to Decompose(p, T) to be performed inside heavy-load calls. Reaching the former goal is discussed after Definition 5 while the latter one is achieved by the *Split-by-height* strategy presented at the end of this section.

DEFINITION 5. *The task $[F, T]$ is* standard *if for all $f \in F$, modulo $Sat(T)$, the polynomial is not constant and its initial is regular w.r.t. $Sat(T)$.*

Estimating the cost of tasks. Assume from now on that every task in Algorithm 1 is standard. When the polynomial p is chosen at Line 6, we know which operations will be performed by the call Decompose(p, U_1). Indeed, if the initial of p is regular w.r.t. $T := U_1$, Line 1 in Algorithm 3 is useless and we know that $f = p$ holds. Let v be $\mathrm{mvar}(p)$. Therefore, two cases arise: either v is algebraic w.r.t. T and GCD$(C_v, p, C_{<v})$ is called and its cost can be estimated (see for instance [12] for complexity estimates); or v is not algebraic w.r.t. T and Extend$(T \cup \{p\})$ is called, leading again to GCD computations with predictable costs.

The *Split-by-height* strategy. Let $[F, E]$ be a task. We introduce a new procedure, called SplitByHeight(F, E), returning a delayed split of $[F, E]$ with the following requirement: If $[G, U]$ is a task returned by SplitByHeight(F, T)

and $|U| = |T|$ holds then $G = \emptyset$ holds. An algorithm for SplitByHeight(F, T) is easily derived from Algorithm 1 and Proposition 2, leading to Algorithm 4 below.

ALGORITHM 4.

Input: *a task* $[F, T]$

Output: *a delayed split of* $[F, T]$ *such that for all output task* $[G, U]$ *either* $|U| > |T|$ *holds, or both* $|U| = |T|$ *and* $G = \emptyset$ *hold.*

SplitByHeight(F, T) == **generate**
1 $R := [[F, T]]$ *# R is a list of tasks*
2 **while** $R \neq [\,]$ **repeat**
3 *choose and remove a task* $[F_1, U_1]$ *from* R
4 $|U_1| > |T| \Longrightarrow$ **yield** $[F_1, U_1]$
5 $F_1 = \emptyset \Longrightarrow$ **yield** $[F_1, U_1]$
6 *choose a polynomial* $p \in F_1$
7 $G_1 := F_1 \setminus \{p\}$
8 $p \equiv 0 \mod Sat(U_1) \Longrightarrow R :=$ cons $([G_1, U_1], R)$
9 **for** $[H, T] \in$ Decompose(p, U_1) **repeat**
10 $R :=$ cons $([G_1 \cup H, T], R)$

Then we derive a new implementation of Triangularize(F, T) based on SplitByHeight and given as Algorithm 5.

ALGORITHM 5.

Input: *a task* $[F, T]$

Output: *regular chains* $T_1, \ldots, T_e$ *solving* $[F, T]$ *in the sense of Definition 2*

Triangularize(F, T) == **generate**
1 $R := [[F, T]]$
2 *# R is a list of tasks*
3 **while** $R \neq [\,]$ **repeat**
4 *choose and remove* $[F_1, U_1] \in R$ *with max priority*
5 $F_1 = \emptyset \Longrightarrow$ **yield** U_1
6 **for** $[H, T] \in$ SplitByHeight(F_1, U_1) **repeat**
7 $R :=$ cons $([H, T], R)$
8 *sort* R *by decreasing priority*

Two benefits are obtained from Algorithm 5 in view of parallelization. Assume that at each iteration of the **while** loop all tasks with maximum priority are executed concurrently. Then, at most n (the number of variables) iterations are needed. Indeed, after each call to SplitByHeight, and thus after each parallel step, the minimum height of a regular chain in any unsolved tasks of R has increased at least by one. Therefore, the depth of the task tree is at most n. Moreover, at each node, with high probability, the work load has increased in a significant manner.

4. PRELIMINARY IMPLEMENTATION AND EXPERIMENTATION

In the previous section, we showed how to create parallel opportunities at a coarse-grained level by making use of modular methods. Then, we introduced different techniques (standard tasks in Definition 5 in order to estimate costs, the *Split-by-height* strategy in order to "factorize" the task tree) so as to limit the irregularity of tasks and thus to avoid cheap computations combined expensive data communications.

In this section, we first briefly introduce the framework based on ALDOR [28] that supports this implementation. Then, we present our dynamic "task farming" parallel scheme and our *Task Pool with Dimension and Rank Guided dynamic scheduling* (`TPDRG`) method, for achieving both load balancing and for removing redundant computing branches at early stages. In the end, we report our experimentation on some well-known problems.

4.1 Implementation scheme

Our preliminary implementation is realized in the high-performance categorical parallel framework reported in [28]. This framework provides a support of multi-processed parallelism in ALDOR on symmetric multiprocessors and multicores. It has mechanisms to support dynamic task management, and offers functions for data communication via shared memory segments for parametric data types such as `SparseMultivariatePolynomial` by serialization. Further more, a sequential implementation [24] of the Triade algorithm has been developed together with the `BasicMath` library for high performance computing. Many of the categories, domains and packages in this sequential implementation (such as polynomial arithmetic, polynomial GCD and resultant over an arbitrary ring) can be reused or extended for our purpose. These provide us qualified support for realizing a preliminary implementation of the parallel algorithm in a reasonable period of time.

As discussed in the previous sections, this component-level parallelization of triangular decompositions is dynamic and irregular. We propose to manage the dynamic tasks by a "task farming" scheme, where a Manager processor distributes tasks to worker processors. The Manager owns an identifier 0, and it also assigns a unique identifier (`TID`) to each task generated at run time. When a task needs to be processed and a processor is available, the Manager will launch a worker (i.e. process) and pass the `TID` as a command line argument to the worker. The worker takes the task's `TID` as its virtual process identifier to guide its communication with the Manager, as described in [28]. When a worker finishes processing an input task, it sends back to the Manager all output unsolved tasks and writes its solved tasks to its standard output.

Our *task pool with dimension and rank guided dynamic scheduling* is depicted in Figure 1. The *task pool* (which can be seen as an implementation of the list R in Algorithms 5) is managed by a *Manager* processor. The Manager first preprocesses the input task $[F, T]$ which generates child tasks. Then, the Manager selects unsolved tasks with maximum priority (See the last paragraph in Section 2.) determined by the dimension information of the tasks, say $Task1.1$, $Task1.2$ and $Task1.3$, and estimates the cost of each of the selected tasks (See Section 3), denoted by $Cost1.1$, $Cost1.2$ and $Cost1.3$, and sorts them decreasingly, say $Cost1.1 \geq Cost1.2 \geq Cost1.3$. Now the manager will launch worker processes if there are processors available and distributes these tasks following this order. Scheduling tasks by the order of decreasing cost aims at obtaining the best trade off between the scheduling overhead and balanced workload [10]. When there is only one task in the selection, the Manager will process it by itself. It will process on its own the tasks with very low estimated cost. (See paragraph on *Estimating the cost of tasks* in Section 3.) The Manager then proceeds to step 2 to receive the results from the workers for remov-

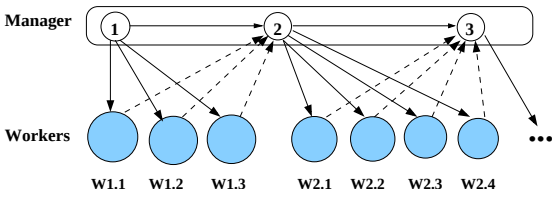

Figure 1: Task pool with dimension and rank guided dynamic scheduling

ing redundant components by inclusion test, and then starts another selection, say the $Task2$'s shown in Figure 1. The overall solving process follows the decreasing order of dimension, which indicates that the dimension of $Task2$'s is lower than the dimension of $Task1$'s. The Manager repeats this scheduling rule until all the tasks are solved. Other than scheduling, the work load of the Manager is very light and it cannot be a bottleneck. The benefits of using standard tasks (to facilitate cost estimates) and solving by decreasing order of dimension have been discussed near the end of Section 3.

To evaluate the effectiveness of our *task pool with dimension and rank guided dynamic scheduling*, we compare below its practical efficiency in our implementation with the *Greedy scheduling* method [16]. In our case, it works as: whenever there is an unsolved task and a free processor, a process is spawned to work on this task. Theoretically, a greedy scheduler is always within a factor of 2 of optimal. However, it cannot ensure the removal of redundant components at an early stage of the solving process.

4.2 Experimental result

Our experimentation was accomplished on Silky in Canada's Shared Hierarchical Academic Research Computing Network (SHARCNET). Silky is a SGI Altix 3700 Bx2 SMP cluster having 128 Itanium2 Processors (1.6GHz). It is a heavy-loaded multiprogrammed computing resource. The system schedules multi-user's job to run. Usually more than 95% memory is in use and almost all the CPUs are used up. This situation does not allow us to test examples that consumes large amount of memory and explains the level of difficulty of our test-examples.

For each problem listed in Table 1, its sequential running time with and without the *regularized initial* condition, imposed by the use of standard tasks (See Definition 5), are listed in column `noregSeq` and column `regSeq` respectively in Table 2. The sequential runs are given by the Triade solver [24]. Column `slowBy` is the ratio between these two timings. This result shows that the cost for maintaining the property of standard tasks is negligible (0.01%). Column `#P` records the number of processors which can give significant speedup to the example's run, that is, beyond it, the increase in the number of processors can not influence significantly its execution time any more. The parallel execution time using this number of processors is recorded in column `SigPar`. The speedup ratio (`SPD`) is calculated by comparing the parallel execution time with respect to the comparable sequential running time `regSeq`.

Table 3 reports on the parallel execution time (wall time) of each problem on a varied number of processors, from 3 to 21. For each run, one processor is always used by the Manager. Thus, given P processors, there are actually $P-1$ which can be scheduled for workers. The corresponding speedups are reported in Table 4.

These results demonstrate that, for these small and medium-sized problems, our component-level (coarse grained) parallel triangular decompositions implemented in a high-level categorical programming language can gain a speedup from 2 to 6, using a considerably small number of processors (from 5 to 9). This is an encouraging result. Unfortunately, this level of parallelization does not show good scalability. For all these small examples, beyond some limit, the speedup cannot increase when adding more processors. The theoretical results by Attardi and Traverso [3] reveal similar nature for coarse grained parallel Gröbner basis computations. For instance, their theoretical speedup of `cyclic7(-last)` is 11.08 by using 136 number of processors. Although our parallelism is very different from theirs in terms of mathematical operations, the performances of component-level parallelism for triangular decompositions also depends on the geometrical property of the input system. The speed-up factor is "essentially" bounded by the number of components with "large degrees" in the output of our modular decompositions. For our Systems 1, 2, 4, 6, this number is clearly 3, 6, 2, 1 respectively, which is close to the corresponding speed-up factors 2.1, 6.1, 2.3 and 1.9. For Systems 5 and 7, which have larger ranges of output component degrees, our claim needs to refined but still gives a good first approximation. System 3 is more subtle: several inconsistent branches explain why we obtain a speed-up of 2.3 with only 1 output component.

In Table 5, for each of the problems, we show the minimum parallel running time (in column `TPDRG`) of our parallel implementation using our *TPDRG* scheduling method and the number of processors used for gaining it, denoted by column `#P (A)`. To evaluate the efficiency of our *TPDRG scheduling*, we also implemented a parallel version using the *Greedy scheduling* for comparison. To reveal the influence of the number of processors, we investigate two timings for the *Greedy scheduling* technique. One is using the same number of processors as used in our best parallel run that we noticed. We record one more to use 2 more processors than that in column `#P (A)`. Except for the very small example *eco6*, all other examples show a better timing for the *TPDRG* scheduling method. This proves that our *TPDRG* scheduling is heuristically efficient. It helps effectively removing redundant components, and hence using less CPU time by avoiding working on redundant tasks. On the contrary, the *Greedy scheduling* cannot ensure removing redundant tasks at an early stage of the solving process.

5. TOWARD EFFICIENT MULTI-LEVEL PARALLELIZATION

We have introduced a component-level parallel algorithm for solving non-linear polynomial systems symbolically by way of triangular decompositions. By using modular methods, we have created opportunities for coarse-grained parallel solving of polynomial systems with rational number coefficients. To exploit these opportunities, we have transformed the Triade algorithm. We have strengthened its notion of a task and replaced the operation Decompose by SplitByHeight in order to reduce the depth of the task tree, create more work at each node, and be able to estimate the cost of each

Sys	*noregSeq* (s)	*regSeq* (s)	*slowBy*	#P	*SigPara* (s)	*SPD*
1	3.63	4.00	0.01	5	1.94	2.1
2	707.53	727.95	0.01	9	119.44	6.1
3	463.02	476.16	0.01	9	207.29	2.3
4	2132.87	2162.40	0.01	9	905.24	2.4
5	4.10	4.14	0.01	9	1.79	2.3
6	866.27	866.20	-	9	455.21	1.9
7	298.33	305.24	0.01	9	96.70	3.2

Table 2: Wall time (s) for sequential (with vs without *regularized initial*) & parallel

#P	Sys 1	Sys 2	Sys 3	Sys 4	Sys 5	Sys 6	Sys 7
3	3.14	355.08	278.70	1401.43	2.10	622.88	104.98
5	1.94	225.29	214.24	1004.69	2.10	481.73	98.44
7	1.91	142.74	209.17	939.40	1.91	470.18	97.19
9	1.91	119.44	207.29	905.25	1.79	455.21	96.70
11	1.95	119.48	207.08	894.27	1.63	453.13	96.38
13	-	119.09	206.38	874.53	1.61	451.93	96.42
17	-	120.01	211.70	865.51	1.63	451.57	96.20
21	-	119.17	-	852.49	-	451.36	96.54

Table 3: Parallel timing (s) vs #processor

#P	Sys1	Sys2	Sys3	Sys4	Sys5	Sys6	Sys7
3	1.3	2.1	1.7	1.5	2.0	1.4	2.9
5	2.1	3.2	2.2	2.2	2.0	1.8	3.1
7	2.1	5.1	2.3	2.3	2.2	1.8	3.1
9	2.1	6.1	2.3	2.4	2.3	1.9	3.2
11	2.0	6.1	2.3	2.4	2.6	1.9	3.2
13	-	6.1	2.3	2.4	2.5	1.9	3.2

Table 4: Speedup vs #processor

System	*TPDRG* (best)	#P (A)	Greedy (A)	#P (B)	Greedy (B)
1	1.91	7	1.79	9	1.78
2	119.09	13	120.51	15	120.52
3	206.38	13	213.21	15	213.35
4	852.49	20	896.79	22	939.62
5	1.61	13	1.63	15	1.63
6	451.36	20	500.50	22	469.35
7	96.20	17	100.78	19	96.17

Table 5: *Best* TPDRG *timing vs* Greedy scheduling (s)

task within each parallel step. This allows us to design a *task pool with dimension and rank guided scheduling* scheme and obtain a heuristically efficient parallelization.

Our preliminary implementation and experimentation demonstrate good performance gain with respect to the comparable sequential solver. We have shown that our multi-processed parallel framework in ALDOR is practically efficient for coarse-grained parallel symbolic computations.

Our long term goal is to achieve an efficient multi-level parallelism: *coarse grained (component)* level for tasks computing geometric objects in the solution sets, and *medium/fine grained* level for polynomial arithmetic such as GCD/resultant computation within each task. We expect that the speedup in the component level would add a multiplicative factor to the speedup of medium/fine grained level parallelization as parallel GCD/resultant computations. Parallel arithmetic for univariate polynomials over fields is well-developed. We need to extend these methods to multivariate case over more general domains with potential of automatic case discussion. A preliminary work in this direction is reported in [26].

6. 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] aldor.org. *The* Aldor *compiler web site*. University of Western Ontario, Canada, 2002.

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

[4] P. Aubry and M. Moreno Maza. Triangular sets for solving polynomial systems: A comparative implementation of four methods. *J. Symb. Comp.*, 28(1-2):125–154, 1999.

[5] E. Becker, T. Mora, M. G. Marinari, and C. Traverso. The shape of the shape lemma. In *Proc. of the international symposium on Symbolic and algebraic computation*, pages 129–133, New York, NY, USA, 1994. ACM Press.

[6] R. Bradford. A parallelization of the Buchberger algorithm. In *Proc. of the international symposium on Symbolic and algebraic computation*, New York, NY, USA, 1990. ACM Press.

[7] R. Bündgen, M. Göbel, and W. Küchlin. A fine-grained parallel completion procedure. In *ISSAC '94: Proceedings of the international symposium on Symbolic and algebraic computation*, pages 269–277, New York, NY, USA, 1994. ACM Press.

[8] 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.

[9] C. Chen, F. Lemaire, M. Moreno Maza, W. Pan, and Y. Xie. Efficient computations of irredundant triangular decompositions with the `regularchains` library. In *Proc. of CASA 2007*, 2006.

[10] J. Coffmann and P. J. Denning. *Operating Systems Theory*. Prentice Hall, 1973.

[11] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM Press, 2005.

[12] 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.

[13] J.-C Faugère. Parallelization of Gröbner bases. In *Proc. PASCO'94*. World Scientific Publishing Company, 1994.

[14] P. Gianni, B. Trager, and G. Zacharias. Gröbner Bases and Primary Decomposition Of Polynomial Ideals. *J. Symb. Comp.*, 6:149–167, 1988.

[15] J. Grabmeier, E. Kaltofen, and V. Weispfenning, editors. *Computer Algebra Handbook*. Springer, 2003.

[16] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(3):416–429, March 1969.

[17] The Computational Mathematics Group. The basicmath library. NAG Ltd, Oxford, UK, 1998. http://www.nag.co.uk/projects/FRISCO.html.

[18] H. Hong and H. W. Loidl. Parallel computation of modular multivariate polynomial resultants on a shared memory machine. In B. Buchberger and J. Volkert, editors, *Proc. of CONPAR 94–VAPP VI,*

Springer LNCS 854., pages 325–336. Springer Verlag, September 1994.

[19] R. D. Jenks and R. S. Sutor. *AXIOM, The Scientific Computation System*. Springer-Verlag, 1992. AXIOM is a trade mark of NAG Ltd, Oxford UK.

[20] M. Kalkbrener. A generalized euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symb. Comp.*, 15:143–167, 1993.

[21] D. Lazard. A new method for solving algebraic systems of positive dimension. *Discr. App. Math*, 33:147–160, 1991.

[22] G. Lecerf. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *J. Complexity*, 19(4):564–596, 2003.

[23] F. Lemaire, M. Moreno Maza, and Y. Xie. The `RegularChains` library. In *Maple 10*, Maplesoft, Canada, 2005. Refereed software.

[24] F. Lemaire, M. Moreno Maza, and Y. Xie. Making a sophisticated symbolic solver available to different communities of users. In *Proc. of* Asian Technology Conference in Mathematics *'06*, 2006.

[25] A. Leykin. On parallel computation of Gröbner bases. In *ICPP Workshops*, pages 160–164, 2004.

[26] X. Li and M. Moreno Maza. Multithreaded parallel implementation of arithmetic operations modulo a triangular set. In *Proc. PASCO'07*, ACM Press, 2007.

[27] Maplesoft. *Maple 10*. http://www.maplesoft.com/, 2005.

[28] M. Moreno Maza, B. Stephenson, S. M. Watt, and Y. Xie. Multiprocessed parallelism support in aldor on smps and multicores. Submitted to PASCO 2007, 2007.

[29] 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. http://www.csd.uwo.ca/∼moreno.

[30] M. Moreno Maza and R. Rioboo. Polynomial gcd computations over towers of algebraic extensions. In *Proc. AAECC-11*, pages 365–382. Springer, 1995.

[31] M. O. Rayes, P. S. Wang, and K. Weber. Parallelization of the sparse modular gcd algorithm for multivariate polynomials on shared memory multiprocessors. In *Proc. of the international symposium on Symbolic and algebraic computation*, pages 66–73, New York, NY, USA, 1994. ACM Press.

[32] T. Shimoyama and K. Yokoyama. Localization and primary decomposition of polynomial ideals. *J. Symb. Comput.*, 22(3):247–277, 1996.

[33] A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM J. Numer. Anal.*, 38(6):2022–2046, 2001.

[34] D. M. Wang. *Elimination Methods*. Springer, Wien, New York, 2000.

[35] V. Weispfenning. Canonical comprehensive grobner bases. In *ISSAC 2002*, pages 270–276. ACM Press, 2002.

[36] W. T. Wu. A zero structure theorem for polynomial equations solving. *MM Research Preprints*, 1:2–12, 1987.

[37] Y.W. Wu, W.D. Liao, D.D. Lin, and P.S. Wang. Local and remote user interface for ELIMINO through OMEI. Technical report, Kent State University, Kent, Ohio, 2003. http://icm.mcs.kent.edu/reports/.

[38] 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.