Solving Polynomial Systems Symbolically and in Parallel

Marc Moreno Maza & Yuzhen Xie,
ORCCA, University of Western Ontario, London, Ontario, Canada
{moreno, yxie}@orcca.on.ca

October 10, 2006

Abstract

We discuss the parallelization of algorithms for solving polynomial systems symbolically. We introduce a *component-level* parallelization: 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 geometry of its solution set. This approach creates new opportunities for parallel execution of polynomial system solvers, in addition to the classical ones at the level of polynomial reduction.

We have developed a parallel framework by extending the ALDOR programming language to support multi-processed parallelism targeting symmetric multiprocessor machines. This allowed us to realize a symbolic polynomial system solver with a component-level parallelization. Our experimentation demonstrates good performance gain with respect to the comparable sequential solver.

1 Introduction

Solving systems of algebraic or differential non-linear polynomials is one of the fundamental problems in mathematical sciences. It has been studied for centuries and has stimulated many research developments. Algorithmic solutions can be classified into three categories: numeric, symbolic and hybrid numeric-symbolic. The choice for one of them depends on the characteristics of the system of equations to solve; for instance, it depends on whether the coefficients are known exactly or are approximations obtained from experimental measurements. This choice depends also on the expected answers, which could be a complete description of all the solutions, or only the real solutions, or just one solution among all of them.

Symbolic solvers 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 [17] 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.

Since the discovery of Gröbner bases [8], the algorithmic advances in symbolic polynomial system solving have made possible to tackle many classical problems that were previously out of reach. However, algorithmic progress is still desirable, for instance when solving symbolically a large system of algebraic nonlinear 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. 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 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 naturally must involve 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 decomposition [16, 29], comprehensive Gröbner bases [33] triangular decompositions [34, 21, 22, 26, 31] 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 speed-up 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 resources consumption 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, and, for instance, by the results of [10] for systems with infinitely many solutions. This phenomenon, however, can be overcome for polynomial systems with coefficients modulo a prime number. Indeed, the probability q_n for a monic uniformly random univariate polynomial of degree n to be irreducible modulo a prime number p with $p^n \geq 16$ satisfies $1/2n \leq q_n \leq 1/n$ [15]; hence a polynomial of large degree irreducible over $\mathbb{Q}$ is likely to factor modulo p.

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. It has been introduced in [27] and has been implemented in the ALDOR language [2] and in the computer algebra systems AXIOM [20] and MAPLE [25] as the `RegularChains` library [23]. 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 [12]. The second 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 2, this allows to execute concurrently the tasks that are the most resources demanding, 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 geometry of its solution set. We do not aim at replacing the previous approaches for parallelizing algorithms. 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, 9, 11, 3, 24]. Second, the Characteristic Set Method of Wu [34], see [1, 35, 36]. 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 [28, 19]. The addition of this second level to the Triade algorithm is work in progress.

We have realized a preliminary implementation in the ALDOR programing language. We use multi-processed parallelism, with interprocess communication through shared memory segments. ALDOR has been designed to express the extremely rich and complex variety of structures and algorithms in computer algebra, with a focus on high-performance computing. However, ALDOR had no mechanism for parallel execution adapted to our needs. In Section 3 we explain how we have enhanced the `BasicMath` library [18] for achieving our goals. We also report on our experiments on a SMP machine.

2 Component-level parallelization

2.1 Incremental solving

Incremental solving is the first idea behind the Triade algorithm for computing triangular decompositions of polynomial systems. Let us give an example before formal definitions. Consider the polynomial system

$$\left\{\begin{array}{l} f_1 = 0 \\ f_2 = 0 \\ f_3 = 0 \end{array}\right. \quad \text{where} \quad \left\{\begin{array}{l} f_1 = x^2 + y + z - 1 \\ f_2 = x + y^2 + z - 1 \\ f_3 = x + y + z^2 - 1 \end{array}\right.$$

and the variable order $x > y > z$. Solving F incrementally means solving successively the subsystems: $f_1 = 0$, then $f_1 = f_2 = 0$ and finally $f_1 = f_2 = f_3 = 0$. The polynomial f_1 is irreducible, thus solving $f_1 = 0$ just produces f_1 as output. Solving $f_1 = f_2 = 0$ leads to eliminate x and produces

$$\left\{\begin{array}{l} x + y^2 + z = 1 \\ y^4 + (2z - 2)y^2 \ + y - z + z^2 = 0 \end{array}\right. .$$

This output system T has a triangular shape, and other algebraic properties: it can be seen as a *solved system*. Solving $f_1 = f_2 = f_3 = 0$ leads to compute the common solutions of T and f_3. To do so, one eliminates x in f_3 obtaining a second polynomial in y and z only. The resultant [15] of these two bivariate polynomials is a univariate polynomial in z, from which the rest of the computations are easily carried out, bringing the following output:

$$\left\{\begin{array}{r} x+y=1 \\ y^2-y=0 \\ z=0 \end{array}\right.$$

$$\left\{\begin{array}{r} 2x+z^2=1 \\ 2y+z^2=1 \\ z^3+z^2-3z=-1. \end{array}\right.$$

There are two components, since two solution points have a null z-coordinate, whereas there is only one solution point for each root z of $z^3+z^2-3z=-1$.

We introduce now the necessary formalism for presenting the properties of the Triade that are favorable to its parallelization. The involved notions of a regular chain and a quasi-component can be found in [4] and are recalled in the Appendix for the reader's convenience.

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 **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 *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:*

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

In particular, for an input polynomial system F the Triade algorithm computes regular chains $T_1, \ldots, T_\ell$ such that we have:

$$V(F) = \cup_{i=1}^{\ell} W(T_i).$$

Informally, this means that the algorithm produces a decomposition of the zero set of the input (unsolved) system F into zero sets of **solved** systems. Indeed, due to its triangular shape, one can view a regular chain as a solved system. (This generalizes the idea that a triangular linear system of equations can be seen as a system in a solved form.)

2.2 Solving by decreasing order of dimension

Solving by decreasing order of dimension is the second idea behind the Triade algorithm. Again we use a simple example to motivate this strategy:

$$\left\{\begin{array}{l} f_1 = x-2+(y-1)^2 \\ f_2 = (x-1)(y-1)+(x-2)y \\ f_3 = (x-1)z \end{array}\right.$$

Factorizing f_3 leads to two sub-systems:

$$S_1 : \left\{\begin{array}{l} y=0 \\ x=1 \end{array}\right. \quad \text{and } S_2 \left\{\begin{array}{l} x-1+y^2-2y=0 \\ (2y-1)x+1-3y=0 \\ z=0 \end{array}\right.$$

The sub-system S_1 is solved. Continuing with the S_2 leads finally to

$$\left\{\begin{array}{l} z=0 \\ y=0 \\ x=1 \end{array}\right., \left\{\begin{array}{l} z=0 \\ y=1 \\ x=2 \end{array}\right. \quad \text{and} \quad \left\{\begin{array}{l} z=0 \\ 2y=3 \\ 4x=7 \end{array}\right.$$

Observe that the leftmost solution point is a special case of the sub-system S_1. Hence, we have exhibited a *redundant component*. All algorithms computing decompositions of polynomial systems have to face this difficulty. In the Triade algorithm, this is approached by generating the output quasi-components (or, equivalently, regular chains) by decreasing order of dimension. This allows to remove the redundant components at an early stage of the computations. Indeed, if the quasi-component $W(T_1)$ is contained in the quasi-component $W(T_2)$, then dimension of T_2 is less or equal to that of T_1, that is $|T_1| \leq |T_2|$ holds, see [27] for details.

Generating quasi-components by decreasing order of dimension leads to difficulty. Due to the incremental solving paradigm, the "basic routine" of the Triade algorithm is the computation of intersections of the form $Z(\{p\}, T)$ (for a polynomial p and a regular chain T). Such an intersection may consist of components of different dimensions, as shown by our second example (the sub-system S_1 has dimension 1 and the sub-system S_2 has dimension 0).

Resolving this conflict of interest between incremental solving and solving by decreasing of order of dimension is achieved by a form of lazy evaluation, formalized below.

2.3 Lazy evaluation

Definition 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:*

(D_1) $Z(F_i, T_i) \prec Z(F, T)$,

(D_2) $Z(F, T) \subseteq Z(F_1, T_1) \cup \cdots \cup Z(F_d, T_d)$,

(D_3) $\mathbf{Sat}(T) \subseteq \mathbf{Sat}(T_i)$,

(D_4) $F_i \neq \emptyset \implies F \subseteq F_i$,

(D_5) $F_i = \emptyset \implies W(T_i) \subseteq V(F)$.

Property (D_1) 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 [26]). Properties (D_2) to (D_5) imply:

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

Proposition 1 tells us that delayed splits can be "composed":

Proposition 1 *If the tasks* $[F_1, T_1], [F_2, T_2]$ *form a delayed split of the task* $[F, T]$ *and the tasks* $[F_{11}, T_{11}]$, $[F_{12}, T_{12}]$ *form a delayed split of the task* $[F_1, T_1]$*, then we have*

$$[F, T] \longmapsto_D [F_{11}, T_{11}], [F_{12}, T_{12}], [F_2, T_2].$$

Proposition 1 allows us to show that solving the task $[F, T]$ reduces to solving tasks

- of the form $[t, T]$ 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, one can design

- an operation extend(t, T) producing a delayed split of the above task $[p, T]$ and,
- an operation 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 $F_i = \emptyset \iff |T_i| = |T \cup \{t\}$. Hence, the output tasks $[F_i, T_i]$ of these operations are solved, i.e. $F_i = \emptyset$ if only and if the dimension of T_i equals that of the input $T \cup \{t\}$. This implies that for each $[F_i, T_i]$ satisfying $|T_i| > |T \cup \{t\}$, the set F_i is not empty, and, thus, the task $[F_i, T_i]$ is not solved. Therefore, the operations extend(t, T) and decompose$(p, T \cup t)$ solve 'completely" in the cases where the dimension of the regular chains T_i remain that of $T \cup \{t\}$ and solve 'lazily" (and, in fact, postpone the computations) in the others.

2.4 Component-level parallelization

We sketch now a procedure solving an input task $[F, T]$ and generating all computed regular chains (final or intermediate) by decreasing order of dimension. This procedure uses three global variables

- 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 decompose applies.

(3) If $V' \neq \emptyset$, then apply decompose once 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 from **(1)** to **(4)** lead naturally to parallel execution. This procedure is also different from Algorithm 2 in [26], which was meant to be executed sequentially.

Generating regular chains by decreasing size has at least two benefits. First, as mentioned above, it allows to detect redundant components (by means of an inclusion test) 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 and load balancing. Indeed, when computing decompose$(p, T \cup t)$ the larger is $|T|$, the more expensive are calculations modulo **Sat**(T). See for instance the complexity results in [13].

2.5 Combing with modular methods

Since it is a decomposition algorithm, the Triade has the potential of parallelization. However, as mentioned in the Introduction, a naive implementation may not be successful with systems over the field $\mathbb{Q}$ of rational numbers. However, the situation changes for polynomial systems with coefficients modulo a prime number. For the 9 examples (all well-known problems from [30]) used in the experiments reported below:

- only one of them has more than one regular chain in its triangular decomposition over $\mathbb{Q}$, (2) but all of them splits in several components when solving modulo a prime number.

For each system, the prime that we use is *large enough* such that the triangular decomposition over $\mathbb{Q}$ can be lifted from the modular one, by means of the techniques introduced in [12]. In addition, for each system, the lifting step consumes much less resources (time and space) than the modular triangular decomposition, as reported in [12].

For each of these systems, Table 1 gives: (1) the number n of variables; (2) the maximum total degree d of a monomial; (3) the prime number p used for the computation of its modular triangular decomposition. In [12], formulas for choosing p are given from n, d and other quantities which can be read easily from the input system.

For each of these systems, Table 1 gives two lists where the i-th item corresponds to the i-th component T_i in the triangular decomposition of the system modulo p. The number of solutions of T_i is found in the first list whereas the output size of T_i is found in the second.

Sys	Name	n	d	p
1	eco6	6	3	105761
2	Weispfenning-94	3	5	7433
3	Issac97	4	2	1549
4	dessin-2	10	2	358079
5	eco7	7	3	387799
6	Methan61	10	2	450367
7	Reimer-4	4	5	55313
8	Uteshev-Bikker	4	3	7841
9	gametwo5	5	4	159223

Table 1: Features of the polynomial systems

Sys	$NumSolutions$	$Size$
1	[1,1,2,4,4,4]	[56,57,721,1205,1293,1283]
2	[2,2,9,35,3,3]	[100,99,282,1048,134,135]
3	[4,7,3,2]	[561,749,458,334]
4	[1,1,6,12,22]	[98,98,885,1100,1448]
5	[1,1,1,1,4,2,4,4,4,4,4,2]	[67,72,73,76,4776,2603,4770,4755,4770,4751,4764,2601]
6	[1,1,1,3,18,3]	[109,105,106,961,2307,957]
7	[1,1,1,1,4,4,24]	[35,35,35,35,350,352,868]
8	[1,1,1,1,2,30]	[16,27,32,27,472,2006]
9	[14,19,11]	[2811,2987,2700]

Table 2: Analysis of the solution sets

One can observe that for each system, the number of solutions and size are rather well balanced between components. This suggests that combining the modular algorithm of [12] and the properties of the Triade algorithm should lead to successful component-level parallelization.

3 Implementation

In the previous section, we showed how to create rich opportunities for a component-level parallel execution, with load balancing. However, another big challenge remains: the implementation. First of all, solving non-linear polynomial systems symbolically by way of triangular decompositions involves sophisticated algorithms and complex mathematical structures. This requires a very high-level programming language. In addition, this language should be suitable for high performance computing, and permits efficient implementation of fast polynomial arithmetic, as discussed in [14]. Secondly, our parallel scheme includes dynamic task management and heavy data communication along with intensive computations. Therefore, the parallel architecture that we run on will significantly influence the implementation scheme and its overall performance.

To meet these challenges effectively, we have developed a parallel framework based on symmetric multiprocessor architecture (SMP and multi-core) by extending the ALDOR programming language to support multi-processed parallelism. We have realized a preliminary implementation of the algorithm presented in Section 2. Tests on benchmark systems have shown a promising performance improvement with respect to the comparable sequential solver.

3.1 A framework for parallel symbolic computations

We choose the ALDOR programming language and symmetric multiprocessor machines (SMP and multi-core) to build our parallel programming model. In our first attempt a multi-processed parallel framework using ALDOR targeting SMPs and multi-cores has been established.

ALDOR has been designed to express the extremely rich and complex variety of structures and algorithms in computer algebra with focuses on interoperability with other languages and high-performance computing. This language has a two-level object model of *categories* and *domains*, that is similar to *interfaces* and *classes* in Java. They provide a type system that allows the programmer the flexibility to extend or build on existing types, or create new categories and domains, as is usually required in algebra.

In addition, an ALDOR program can be compiled into: stand-alone executable programs; object libraries in native operating system formats (which can be linked with one another, or with C or Fortran code to form application programs); portable byte code libraries; and C or Lisp source [7]. Aggressive code optimizations by techniques such as program specialization, cross-file procedural integration and data structure elimination, are performed at intermediate stages of compilation [32]. This produces code that is comparable to hand-optimized C.

For these reasons a sequential implementation of the Triade algorithm in ALDOR has been developed together with the `BasicMath` library for high performance computing. Many of the categories, domains and packages of this sequential implementation (polynomial arithmetic, polynomial GCD and resultant computation, inclusion test for components) 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.

Our implementation aims at efficiently using multiprocessors with shared memory to gain best practical efficiency. Indeed, they will have significant effects to reduce the parallel overhead for applications like ours, involving dynamic task management and heavy data communication among the processors.

Before our work, however, like many other computer algebra systems, ALDOR programming language did not have any support for parallel programming, not to mention MPI binding, or OpenMP binding. Fortunately, an ALDOR program can be compiled into stand-alone executable programs. This allows us to build separated executable modules to run as independent parallel processes. ALDOR also has a primer `exec()`, for initiating a program `P` from within a program `Q`. Unlike `fork()` in C, where a program can be executed in a child process, which is an identical copy of the parent process, programs `P` and `Q` do not have any relationships. In another word, they are executed as two independent processes. Anyhow, these give us the basic functionalities for dynamic task management.

Now comes the challenge of establishing interprocess communication (IPC) in ALDOR, which is critical for all parallel programming environment. We rely on *shared memory segments* for System V IPC and develop a domain called `SharedMemorySegment` in ALDOR, based on the interoperability of ALDOR with the C programing language. The `SharedMemorySegment` domain has methods for creating a segment and connecting to it, attaching i.e. getting a pointer to the segment, reading and writing, and detaching from and deleting the segment. An element of the domain `SharedMemorySegment` can be either a string or a primitive array of machine integers.

There are mainly two advantages to use shared memory segment for interprocess data communication in ALDOR. One is that it is an efficient way for System V IPC. Secondly it is suitable for large data communication. For a given operating system, the maximum size of the memory that is available for shared memory segments is set by default, but it can be modified. It is also worth to point out that our domain `SharedMemorySegment` was designed to handle only primitive data types (strings, primitive arrays of machine integers) in ALDOR. This provides a unified way of interprocess data communication between ALDOR processes. Of course, this is not a limitation since an object of any other type can always be converted into a primitive array of machine integers.

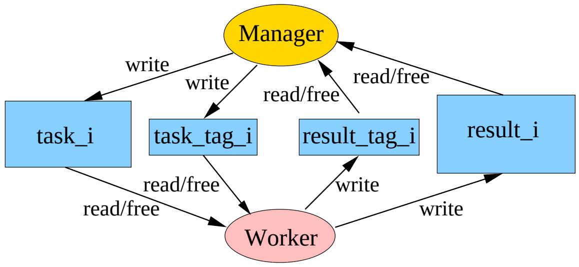
Another difficult issue for data communication through this heterogeneous environment is the many complex data types in our program. For example, sparse multivariate polynomial and dense multivariate polynomial are all valid types in our program in ALDOR. In our package, a Triade task (see Definition 1) is a list of lists of polynomials with special properties (including the list of variables and the ring characteristic). It can be described by $[F, T]$, where, F is a list of unsolved polynomials; and, T is a list of polynomials forming a regular chain.

To achieve effective data communication, we convert a dense multivariate polynomial into a primitive array of machine integers via a univariate polynomial by means of Kronecker substitution [15]. We convert a sparse multivariate polynomial into a distributed multivariate polynomial, which is represented by a tree of terms, where a term is an exponent vector together with a coefficient; next we traverse this tree to get a primitive array of machine integers. This latter representation is more compact and occupies less memory space than the former one.

3.2 Implementation scheme and synchronization

Another main concern is the concurrency (synchronization) control for multiprocessing in ALDOR, since there was no such mechanisms before. For our component-level parallel solving reported in Section 2, we apply a “process farm” parallel scheme consisting of a *Manager* process and *worker* processes initiated by the Manager when needed. A worker executes either a decompose or a extend operation (see the previous section for these operations) and then terminates itself. The Manager distributes tasks to and collects results from the worker processes. This allows the Manager to perform the removal of the redundant components. The Manager maintains a task table and assigns a unique ID to each of the tasks generated by the workers.

Between the Manager and a worker process,



Synchronizing Manager with a worker

data communication is synchronized by four shared memory segments defined by a protocol related to the task ID. Let the task ID be i. The four segments are named as $task\_i$, $task\_tag\_i$, $result\_tag\_i$, and $result\_i$ respectively. The main strategy is described in the above picture. At a time the Manager process selects tasks with highest priority for processing. Let tasks with ID of i, j, k be chosen. The manager first writes the task with $ID = i$ into a shared memory segment named $task\_i$, then writes the size of this task (the length of the integer array) into another segment named $task\_tag\_i$. Now it launches a worker and passes the value i (the task ID) as a command line argument to the worker process. Continually, the Manager process will do the same to task j and task k. Then for tasks i, j, k, the Manager will check in turn their result tag segments named $result\_tag\_i$, $result\_tag\_j$ and $result\_tag\_k$. If there is a result in any one of the result tag segments, for instance j, it will read the size of the result for task j from this segment named $result\_tag\_j$, and free it. Now the Manager knows the result of task j is returned and its size, then it will read the result for task j from the segment named $result\_j$, and free this shared memory segment at the end. The same operation will be done to both task i and task k.

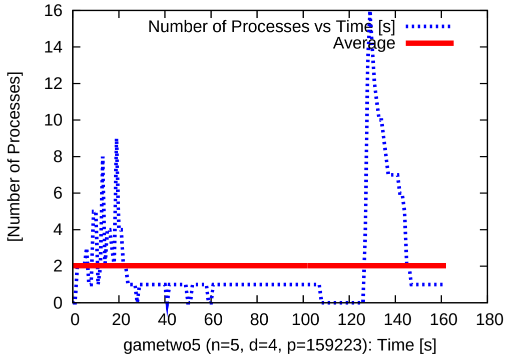
When a worker is launched, it will get the task ID from the command line argument. Let this ID be m. By the protocol, the worker will know the names of the four shared memory segments to work with, and the order and permission to access them. It will first read the size of the task from the shared memory segment named $task\_tag\_m$, and then read the task from the segment named $task\_m$. Now it can free the above two segments. When finishing the *decompose* operation on this task, the worker will write first the result into the segment named $result\_m$, then the size of the result into the segment named $result\_tag\_m$. Then, it terminates by itself.

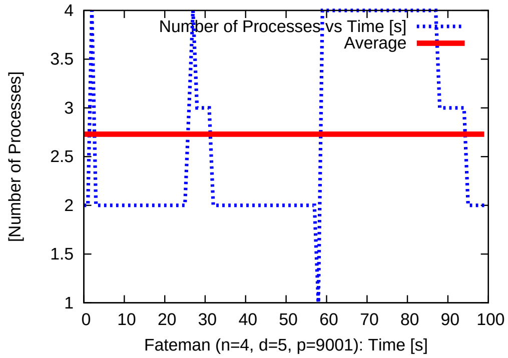
3.3 Experimentation

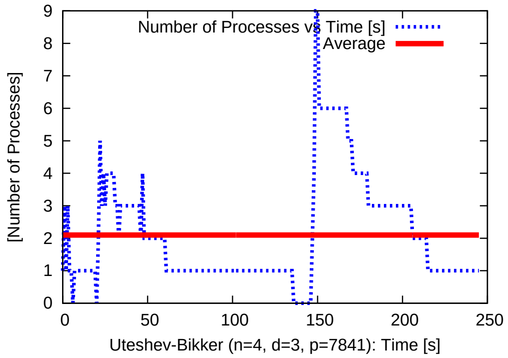
We realized a preliminary implementation of our component-level parallel solving of nonlinear polynomial systems symbolically based on the above framework. Our experimentation was done on a symmetric multiprocessor, AuthenticAMD with four CPUs (AMD Opteron(tm) Processor 850, 2390MHz) and 32-bit 8 GB total memory. The version of Linux we are running is Fedora Core release 3. The maximum amount of memory that can be allocated for shared memory is 33 MB.

The polynomial systems that are used in this experimentation are taken from [30]. 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 three examples, namely: `gametwo5`, `Uteshev-Bikker`, and `Fateman`. 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 these 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 component-level parallel execution, 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.

4 Discussion and conclusion

We introduce a component-level parallel algorithm for solving non-linear polynomial systems symbolically by way of triangular decompositions. This algorithm employs techniques of solving by decreasing order of dimension, lazy evaluation, and modular methods to obtain rich parallel opportunities and load balancing. We have developed a parallel framework by extending the ALDOR programming language to support multi-processed parallelism targeting symmetric multiprocessor machines. This framework can benefit to other algorithms in adapting to the new parallel architectures, such as SMPs and multi-cores. Our experimentation demonstrates good performance gain with respect to the comparable sequential solver. We are currently scaling to a 64 bits 128-processor SMP in Canada's Shared Hierarchical Academic Research Computing Network (SHARCNET).

Through this work we have noticed that the heavy overhead from dynamic process management and data communication is a bottleneck for an efficient parallel execution. In fact, this is a big challenge for parallel symbolic computations in general. Our next objective is to build threads in ALDOR to support multi-threaded parallelism for symbolic computations targeting SMP and multi-cores. Threads of the same process have much less overhead and synchronize data much quicker. The emerging multiprocessor machines (SMPs and multi-cores) fully support thread-level parallelism. In particular, these multi-cores with direct connect architecture will greatly improve the performance of our solver if we use multi-threaded parallelism. The ALDOR's interoperability with C and the machine resources provides a feasible way to construct ALDOR threads. Special concerns will be on the effective means to handle the generic types for ALDOR threads, such as polynomial data types, so that we can hide their internal complexity and provide an ease-of-use framework for general users.

In addition, we have recognized that we can gain higher scalability and more parallel opportunities by applying medium and fine grained parallelization for polynomial arithmetic such as multiplication, GCD/resultant, and factorization involved in each task. 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.

5 Appendix

Let $\mathbb{K}$ be a field and $X = x_1 < \cdots < x_n$ be ordered variables. Let $\overline{\mathbb{K}}$ be an algebraically closed field containing $\mathbb{K}$. Usually $\mathbb{K} = \mathbb{Q}$, the field of rational numbers and $\overline{\mathbb{K}} = \mathbb{C}$, the field of complex numbers.

For a non-constant polynomial $p \in \mathbb{K}[X]$, the *main variable* of p, denoted by $\mathsf{mvar}(p)$, is the greatest variable of p; the *initial* of p, denoted by $\mathsf{init}(p)$ is the leading coefficient of p w.r.t. $\mathsf{mvar}(p)$. For example, for $p_1 = (2x_1 - 1){x_2}^2 - 3x_1x_2 + x_2$, $\mathsf{mvar}(p_1) = x_2$, and $\mathsf{init}(p_1) = 2x_1 - 1$.

Let $F \subset \mathbb{K}[X]$ be any set of polynomials with coefficients in $\mathbb{K}$ and variables in X. We denote by $\langle F \rangle$ the ideal generated by F in $\mathbb{K}[X]$ and by $\sqrt{\langle F \rangle}$ its radical. We denote by $V(F)$ the *zero set* or *algebraic variety* of F in the affine space $\overline{\mathbb{K}}^n$, that is the set of the common zeros of the polynomials of F.

It is important to observe that not every subset of $\overline{\mathbb{K}}^n$ is the zero set of some subset of $\mathbb{K}[X]$. Hence, the following topological notion plays a central role. 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 simply the intersection of the $V(F)$ containing W, for all $F \subset \mathbb{K}[X]$.

Let $\mathcal{I}$ be a proper ideal of $\mathbb{K}[X]$. We say that a polynomial $p \in \mathbb{K}[X]$ is *regular* modulo $\mathcal{I}$ if p is neither null modulo $\mathcal{I}$, nor a zero-divisor modulo $\mathcal{I}$.

Let $h \in \mathbb{K}[X]$. We denote by $\mathcal{I} : h^\infty$ the set of the polynomials p such that there exists a non-negative integer e such that $h^e\, p$ belongs to $\mathcal{I}$. Informally, an element of $\mathcal{I} : h^\infty$ can be seen as a fraction with numerator in $\mathcal{I}$ and whose denominator is a power of h.

Let $T = t_1, \ldots, t_s$ be non-constant polynomials in $\mathbb{K}[X]$ with respective (pairwise distinct) main variables $\mathsf{mvar}(t_1) < \cdots < \mathsf{mvar}(t_s)$. The *saturated ideal* of T is defined by

$$\mathbf{Sat}(T) = \langle T \rangle : {h_T}^\infty,$$

where h_T is the product of the initials of the polynomials of T.

The *quasi-component* of T is the of the zero-set $V(T)$ consisting of all the points that do not cancel any of the initials of the polynomials of T. In other words, the $W(T)$ is the system of equations and inequations:

$$t_1 = 0, \ldots, t_s = 0, h_1 \neq 0, \ldots, h_s \neq 0$$

where h_i is the initial of t_i for $i = 1 \cdots s$. The subset $W(T)$ is not necessarily the zero set of some subset of $\mathbb{K}[X]$. With $\mathbb{K} = \mathbb{Q}$, $n = 2$ and $T = \{x_1x_2\}$, the quasi-component of T consists of the complex line minus the origin.

We have the following important property, which realizes a bridge between the "algebraic" notion of a saturated ideal and the "geometric" notion of a quasi-component. It is, in fact, a consequence of Hilbert's theorem of zeros:

$$\overline{W(T)} = V(\mathbf{Sat}(T)).$$

We can now define the two central concepts in the theory of triangular decompositions.

Definition 3 *the set T is a* regular chain *if for all $i = 2 \cdots s$ the initial of t_i is regular modulo the saturated ideal of $t_1, \ldots, t_{i-1}$.*

Definition 4 *Let $F \subset \mathbb{K}[X]$ be a polynomial set. A set $C_1, \ldots, C_s$ of regular chains in $\mathbb{K}[X]$*

$$V(F) = \bigcup_{1 \leq i \leq s} W(C_i).$$

As an example, we consider the polynomial system F below with two polynomials and with ordered variables $x > y > a > b > c > d > e > f$:

$$\left\{ \begin{array}{lcl} ax + cy - e & = & 0 \\ bx + dy - f & = & 0 \end{array} \right.$$

The polynomial set T below is a regular chain such that we have $W(T) \subset V(T)$. This inclusion can be checked by substituing the expressions of x and y given by T into F.

$$\left\{ \begin{array}{l} bx + dy - f \\ (da - cb)\, y - fa + eb \end{array} \right.$$

This substitution would require the initials b and $da - cb$ to be non-zero. The solutions of F for which one of these initials vanishes are given by the regular chains (as one can check again by substitution). Together with T, the eleven regular chains below form a triangular decomposition of F.

$$\left\{ \begin{array}{l} bx + dy - f \\ (da - cb)\, y - fa + eb \end{array} \right. , \left\{ \begin{array}{l} ax + cy - e \\ dy - f \\ b \end{array} \right. ,$$

$$\left\{ \begin{array}{l} bx + dy - f \\ da - cb \\ fc - ed \end{array} \right. , \left\{ \begin{array}{l} dy - f \\ a \\ b \\ fc - ed \end{array} \right. ,$$

$$\left\{ \begin{array}{l} bx - f \\ fa - eb \\ c \\ d \end{array} \right. , \left\{ \begin{array}{l} ax + cy - e \\ b \\ d \\ f \end{array} \right. , \left\{ \begin{array}{l} bx + dy \\ da - cb \\ e \\ f \end{array} \right. ,$$

$$\left\{ \begin{array}{l} cy - e \\ a \\ b \\ d \\ f \end{array} \right. , \left\{ \begin{array}{l} y \\ a \\ b \\ e \\ f \end{array} \right. , \left\{ \begin{array}{l} x \\ c \\ d \\ e \\ f \end{array} \right. , \left\{ \begin{array}{l} a \\ b \\ c \\ d \\ e \\ f \end{array} \right. .$$

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, D. Lazard, and M. Moreno Maza. On the theories of triangular sets. *J. Symb. Comp.*, 28(1-2):105–124, 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] Peter A. Broadbery, Samuel S. Dooley, Pietro Iglio, Scott C. Morisson, Jonathan M. Steinbach, Robert S. Sutor, and Stephen M. Watt. *AXIOM Library Compiler User Guide*. NAG, The Numerical Algorithms Group Limited, Oxford, United Kingdom, 1st edition, November 1994. AXIOM is a registred trade mark of NAG.

[8] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[9] 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.

[10] L. Busé and C. D'Andrea. On the irreducibility of multivariate subresultants. *C. R. Math. Acad. Sci. Paris*, 338(4):287–290, 2004.

[11] 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.

[12] 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.

[13] 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.

[14] A. Filatei, X. Li, M. Moreno Maza, and É Schost. Implementation techniques for fast polynomial arithmetic in a high-level programming environment. In *Proc. ISSAC'06*. ACM Press, 2006.

[15] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

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

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

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

[19] 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.

[20] 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.

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

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

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

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

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

[26] 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.

[27] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. http://www.csd.uwo.ca/∼moreno.

[28] 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.

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

[30] *The SymbolicData Project*. http://www.SymbolicData.org, 2000–2006.

[31] D. M. Wang. *Elimination Methods*. Springer, Wein, New York, 2000.

[32] Stephen M. Watt, Peter A. Broadbery, Samuel S. Dooley, Pietro Iglio, Scott C. Morrison, Jonathan M. Steinbach, and Robert S. Sutor. A first report on the a# compiler. In *ISSAC '94: Proceedings of the international symposium on Symbolic and algebraic computation*, New York, NY, USA, 1994. ACM Press.

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

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

[35] 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/.

[36] 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.