The Design and Implementation of a High-Performance Polynomial System Solver

Alexander Brandt
Supervisor: Marc Moreno Maza

PhD Public Lecture

Department of Computer Science
University of Western Ontario

August 2, 2022

Solving Systems of Equations

Find values of x, y, z which satisfy $$F = \begin{cases} a(x,y,z) = 0 \\ b(x,y,z) = 0 \\ c(x,y,z) = 0 \end{cases}$$

- Solving systems of equations is a fundamental problem in scientific computing
- Numerical methods are very efficient and useful in practice, but only find approximate solutions as floating point numbers
  - ↳ Newton's method, Homotopy methods, Gradient descent
- **Symbolic methods** to find exact solutions are required in robotics, celestial mechanics, cryptography, signal processing [13]
  - ↳ Particularly used to find a complete description of all solutions

Solving a Linear System of Equations

$$\begin{cases} x + 3y - 2z = 6 \\ 3x + 5y + 6z = 7 \\ 2x + 4y + 3z = 8 \end{cases}$$

Step 1: **triangularization**

(a) by *elimination of variables*:

$$\begin{cases} x + 3y - 2z = 6 \\ 3x + 5y + 6z = 7 \\ 2x + 4y + 3z = 8 \end{cases} \xrightarrow[\text{substitute } x]{\text{solve for } x} \begin{cases} x = 5 - 3y + 2z \\ -4y + 12z = -8 \\ -2y + 7z = -2 \end{cases} \xrightarrow[\text{substitute } y]{\text{solve for } y} \begin{cases} x = 5 + 2z - 3y \\ y = 2 + 3z \\ z = 2 \end{cases}$$

(b) by *Gaussian elimination*:

$$\left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 3 & 5 & 6 & 7 \\ 2 & 4 & 3 & 8 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 0 & 1 & -3 & 2 \\ 0 & -2 & 7 & -2 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 0 & 1 & -3 & 2 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

Step 2: **back-substitution** to find particular values for x, y, z

Solving a Non-Linear System of Equations

Via **Gröbner Basis** we can "solve" a non-linear system

$$\begin{cases} x^2 + y + z = 1 \\ x + y^2 + z = 1 \\ x + y + z^2 = 1 \end{cases} \Longrightarrow \begin{cases} x + y + z^2 = 1 \\ (y + z - 1)(y - z) = 0 \\ z^2\left(z^2 + 2y - 1\right) = 0 \\ z^2\left(z^2 + 2z - 1\right)(z - 1)^2 = 0 \end{cases}$$

"Solving" a system is not just about finding particular values, rather:

> *"find a description of the solutions from which we can easily extract relevant data"*

Why?

- A **positive-dimensional system** has *infinitely many solutions*
- *Underdetermined* linear systems, and most non-linear systems
- Univariate polynomials of degree > 4, it may not be possible to have their solutions described in radicals

Decomposing a Non-Linear System

Many ways to “solve” a system

$$\begin{cases} x^2 + y + z = 1 \\ x + y^2 + z = 1 \\ x + y + z^2 = 1 \end{cases} \xRightarrow{\text{Gröbner Basis}} \begin{cases} x + y + z^2 = 1 \\ (y + z - 1)(y - z) = 0 \\ z^2\left(z^2 + 2y - 1\right) = 0 \\ z^2\left(z^2 + 2z - 1\right)(z - 1)^2 = 0 \end{cases}$$

$\Downarrow$ **Triangular Decomposition**

$$\begin{cases} x - z = 0 \\ y - z = 0 \\ z^2 + 2z - 1 = 0 \end{cases}, \quad \begin{cases} x = 0 \\ y = 0 \\ z - 1 = 0 \end{cases}, \quad \begin{cases} x = 0 \\ y - 1 = 0 \\ z = 0 \end{cases}, \quad \begin{cases} x - 1 = 0 \\ y = 0 \\ z = 0 \end{cases}$$

Both solutions are equivalent (via a union)

- by using triangular decomposition, **multiple components** are found, suggesting possible **component-level parallelism**

Research Themes

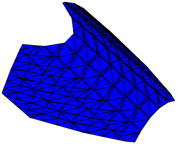
Solving equations is a fundamental computational problem. Triangular decomposition is a core operation in general computer algebra routines (`solve` in *Maple*).

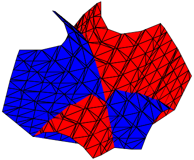
1. Provide algorithmic schemes and implementation techniques for **high-performance polynomial system solvers**
   ↳ Implementations of triangular decomposition are not as sophisticated as those based on Gröbner bases
2. Explore **high-level, irregular parallelism** in symbolic computation
   ↳ Typically limited to low-level, regular parallelism (e.g. arithmetic)
3. Examine **software design** for accessibility and maintainability of **high-performance mathematical software**
   ↳ Re-use, maintainability, and adaptability often missing

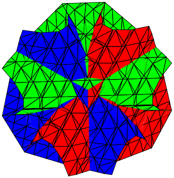
Outline

Incremental Decomposition of a Non-Linear System

Intersect one equation at a time with the current solution set







$$F = \begin{cases} x^2 + y + z = 1 \\ x + y^2 + z = 1 \\ x + y + z^2 = 1 \end{cases}$$

$$\varnothing$$

$$F[1] \quad \downarrow$$

$$\left\{ x^2 + y + z = 1 \right\}$$

$$F[2] \quad \downarrow$$

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

$$F[3] \quad \swarrow \quad \swarrow \quad \searrow \quad \searrow$$

$$\begin{cases} x - z = 0 \\ y - z = 0 \\ z^2 + 2z - 1 = 0 \end{cases}, \quad \begin{cases} x = 0 \\ y = 0 \\ z - 1 = 0 \end{cases}, \quad \begin{cases} x = 0 \\ y - 1 = 0 \\ z = 0 \end{cases}, \quad \begin{cases} x - 1 = 0 \\ y = 0 \\ z = 0 \end{cases}$$

Motivations and Challenges

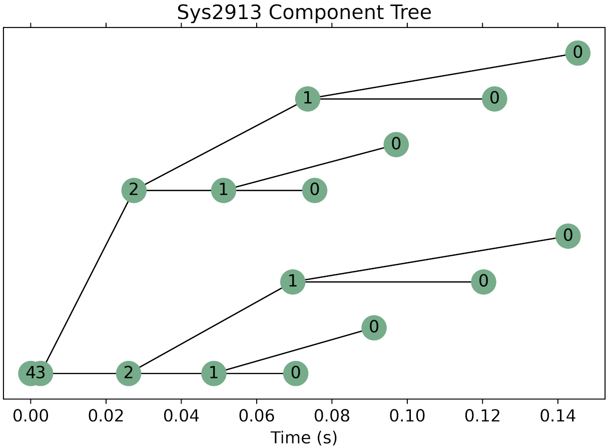
Motivations:

- Symbolic solving is difficult but still desirable in many fields
- Algorithmic development has come a long way [7]; must now focus on implementation techniques, making the most of modern hardware
  - ↳ Multicore processors, cache hierarchy
  - ↳ Must apply **parallel computing** and **data locality**

Challenges:

- The application of high-performance techniques to high-level geometric algorithms
- Different problem instances have different "hot spots": pseudo-division, subresultants, factorization, GCDs, etc.
- Potential **parallelism is problem-dependent** and *not* algorithmic
  - ↳ Geometry may or may not split into different components
  - ↳ Finding splittings is as difficulty as solving the problem

Unbalanced and Irregular Parallelism



- More parallelism exposed as more components found,
- Work unbalanced between branches; this is **irregular parallelism**
- Mechanism needed for **adaptive, dynamic parallelism**

Previous Works

- Long history of theoretical and algorithmic development in triangular decomposition [3, 5, 7–9, 19, 22, 23]

- Parallelization of high-level algebraic and geometric algorithms was more common roughly 30 years ago
  - ↳ Such as in Gröbner Bases [2, 6, 11] and CAD [21]

- Recent parallelism of *low-level* routines with *regular parallelism*:
  - ↳ Polynomial arithmetic [12, 16]
  - ↳ Modular methods for GCDs and Factorization [14, 18]

- High-level computer algebra algorithms, often with *irregular parallelism*, have seen little progress in research or implementation
  - ↳ The normalization algorithm of [4] finds components serially, then processes each component with a simple parallel map
  - ↳ Early work on parallel triangular decomposition was limited by symmetric multi-processing and inter-process communication [20]

Outline

Contributions in this Thesis

1. Algebraic Class Hierarchy
2. Object-Oriented Parallel Support
3. High-Performance Triangular Decomposition
4. Designing the Next Generation of Triangular Decomposition
5. Lazy & Parallel Hensel Factorization

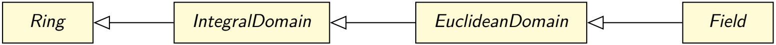
BPAS Library

Basic Polynomial Algebra Subprograms

- An open-source C/C++ library for polynomial algebra
  - ↳ Univariate, bivariate, multivariate polynomials over $\mathbb{Z}, \mathbb{Q}, \mathbb{Z}/p\mathbb{Z}, \mathbb{C}$
  - ↳ GCDs, Factorization, (multi-dimensional) FFTs, Symbolic integration
  - ↳ Triangular decomposition, Hensel factorization
- High-performance implementations for modern architectures: data locality, parallelism
- Over 600,000 lines of code.
- Encapsulate complexity for ease-of-use, maintainability, extensibility

Algebraic Class Hierarchy

Compile-time introspection, **Template Metaprogramming**,



- Ring-like algebraic structures naturally form a hierarchy, but elements of different Rings may not be mathematically compatible
- **Static polymorphism**, **implicit conversion** ensures compile-time mathematical type safety
- Other libraries like *Singular*, *CoCoA*, *LinBox* use run-time values to check compatibility

"Dynamic" type creation

- Creation of new types from composition of others
- Given R, is $R[x]$ a ring? integral domain? Euclidean domain?
- **Conditional Export**: modify interface of `Type<T>` based on `T`

Object-Oriented and Cooperative Parallelism

- Motivated by **dynamic multithreading concurrency platforms**
  - ↳ Cilk, OpenMP, TBB
  - ↳ User specifies *where* concurrency is possible
  - ↳ Runtime decides *what* and *how* to execute in parallel
- Framework entirely encapsulates parallel computing constructs:
  - ↳ Clean user-code
  - ↳ Allows for dynamic multithreading
- Support for **parallel patterns**: meta-algorithms for efficient parallel computing
- Composition and Cooperation of parallel regions:
  - ↳ Layers of parallelism allow for dynamic load-balancing via dynamic resource distribution supports **irregular parallelism**
  - ↳ *Priority tasks*

High-Performance Triangular Decomposition

1. High-performance triangular decomp., core operations in C/C++
2. Cooperative **component-level parallelism** and low-level parallelism
3. Large-scale and systematic experimentation of triangular decomposition

Next-Generation Triangular Decomposition

1. Modular algorithms to avoid expression swell
2. Advances in parallel multivariate polynomial multiplication
3. Algorithms and data structures to avoid redundant computations
   - ↳ **Speculative** subresultants avoids unnecessary computation
   - ↳ **Regular chain universe**

Lazy & Parallel Hensel Factorization

Towards computing limit points, an efficient implementation of EHC, multivariate power series, Laurent series, Puiseux series.

1. **Hensel factorization** via Weierstrass Preparation Theorem
   - ↳ Computes roots of $F(X_1, \dots, X_n, Y)$ as power series in $X_1, \dots, X_n$
2. High-performance, **lazy, multivariate power series**
   - ↳ First known implementation in a compiled code
   - ↳ A basis toward Laurent series and Puiseux series
3. Complexity analyses for Hensel factorization, WPT
4. **Parallel pipeline** implementation of Hensel factorization to compute all roots simultaneously
   - ↳ First known pipeline implementation in symbolic computation

Hensel's Lemma: A Brief Overview

An approximate factorization can be "lifted" to the true factorization

1. **The Polynomial Case**
   - ↳ $F(X_1, \ldots, X_n, Y) = F(\underline{X}, Y) = f_1\, f_2 \cdots f_r$, f_i are polynomials
   - ↳ Given upper bounds on the degs. of f_i: evaluation-interpolation
   - ↳ Over $\mathbb{Z}_p[X, Y]$, Hensel lifting can be done in $\mathcal{O}({d_X}^2 d_Y + d_X {d_Y}^2)$ [17]

Assume F is squarefree;

Hensel's Lemma: A Brief Overview

An approximate factorization can be "lifted" to the true factorization

1. **The Polynomial Case**
   - ↳ $F(X_1, \ldots, X_n, Y) = F(\underline{X}, Y) = f_1\, f_2 \cdots f_r,\ f_i$ are polynomials
   - ↳ Given upper bounds on the degs. of f_i: evaluation-interpolation
   - ↳ Over $\mathbb{Z}_p[X, Y]$, Hensel lifting can be done in $\mathcal{O}({d_X}^2 d_Y + d_X {d_Y}^2)$ [17]

2. **Polynomials with Puiseux series roots**, k is num. terms in series
   - ↳ *Newton-Puiseux Theorem*: for $F \in \mathbb{C}[X, Y]$, $\mathcal{O}(d^2 M(k))$ [15]
     $F(X, Y) = (Y - f_1) \cdots (Y - f_r),\ f_i$ are Puiseux series in X
   - ↳ *Extended Hensel Construction*: for $F \in \mathbb{K}[X, Y]$, $\mathcal{O}(k^2 d M(d))$ [1]
     $F(\underline{X}, Y) = (Y - f_1) \cdots (Y - f_r),\ f_i$ are Puiseux series in $\underline{X}$

Assume F is squarefree; $M(n)$ is the time required to multiply two polynomials of degree n; $\mathbb{K}$ is algebraically closed

Hensel's Lemma: A Brief Overview

An approximate factorization can be "lifted" to the true factorization

1. **The Polynomial Case**
   - ↳ $F(X_1, \dots, X_n, Y) = F(\underline{X}, Y) = f_1 f_2 \cdots f_r$, f_i are polynomials
   - ↳ Given upper bounds on the degs. of f_i: evaluation-interpolation
   - ↳ Over $\mathbb{Z}_p[X, Y]$, Hensel lifting can be done in $\mathcal{O}({d_X}^2 d_Y + d_X {d_Y}^2)$ [17]
2. **Polynomials with Puiseux series roots**, k is num. terms in series
   - ↳ *Newton-Puiseux Theorem*: for $F \in \mathbb{C}[X, Y]$, $\mathcal{O}(d^2 M(k))$ [15]
     $F(X, Y) = (Y - f_1) \cdots (Y - f_r)$, f_i are Puiseux series in X
   - ↳ *Extended Hensel Construction*: for $F \in \mathbb{K}[X, Y]$, $\mathcal{O}(k^2 d M(d))$ [1]
     $F(\underline{X}, Y) = (Y - f_1) \cdots (Y - f_r)$, f_i are Puiseux series in $\underline{X}$
3. **Polynomials with Power Series Coefficients**
   - ↳ *EHC*: in theory (not implemented) factors polys with power series coefs
   - ↳ **Our solution**: $F = (Y - f_1) \cdots (Y - f_r)$, f_i are power series in $\underline{X}$
     Over $\mathbb{K}[[X]][Y]$: $\mathcal{O}(d_Y^2 k^2)$

Assume F is squarefree; $M(n)$ is the time required to multiply two polynomials of degree n; $\mathbb{K}$ is algebraically closed

Outline

Polynomial Notations

- Let $\mathbb{K}$ be a perfect field (e.g. $\mathbb{Q}$ or $\mathbb{C}$) and $\overline{\mathbb{K}}$ its algebraic closure
- Let $\mathbb{K}[\underline{X}]$ be the set of multivariate polynomials (a *polynomial ring*) with n ordered variables, $\underline{X} = X_1 < \cdots < X_n$.
- For $p \in \mathbb{K}[\underline{X}]$:
  - ↳ the main variable of p is the maximum variable with positive degree
  - ↳ the initial of p is the leading coeff. of p with respect to its main variable
  - ↳ the tail of p is the terms leftover after setting its initial to 0

$$(2y + ba)x^2 + (by)x + a^2 \quad \in \ \mathbb{Q}[b < a < y < x]$$

- The **zero set** of $F \subset \mathbb{K}[\underline{X}]$ is an **algebraic variety**—the geometric representation of its solutions
  - ↳ $V(F) = \left\{(a_1, \ldots, a_n) \in \overline{\mathbb{K}}^n \mid f(a_1, \ldots, a_n) = 0, \ \forall f \in F\right\}$
- For any subset $S \subset \overline{\mathbb{K}}^n$, its **Zariski closure** $\overline{S}$ is the smallest algebraic variety containing S.

Triangular Sets and Regular Chains

A **triangular set** $T \subset \mathbb{K}[\underline{X}]$ is a collection of polynomials with pairwise different main variables

$$T = \left\{ \begin{array}{l} T_v = h\,v^d + \mathrm{tail}(T_v) \\ \\ T_v^- = \left\{ \quad \right\} \end{array} \right\} \subset \mathbb{K}[\underline{X}]$$

Example:

$$T = \left\{ \begin{array}{r} (2y + ba)x - by + a^2 \\ 2y^2 - by - a^2 \\ a + b \end{array} \right\} \subset \mathbb{Q}[b < a < y < x]$$

A triangular set is a **regular chain** if:

(i) T_v^- is a regular chain, and

(ii) h (i.e. $\mathrm{init}(T_v)$) is **regular** (neither 0 nor a *zero-divisor*) w.r.t. T_v^-

The *dimension* of a regular chain T is $n - |T|$.

The foundation of splitting: regularity testing

To intersect a polynomial with an existing regular chain, it must have a regular initial, regularizing finds splittings via a **case discussion**

- either the initial is regular, or it is not regular

$$f = (y+1)x^2 - x$$

$$\begin{tikzcd}
 & T_1 = \begin{cases} y+1=0 \\ z-1=0 \end{cases} \arrow[r, "f = x"] & T_3 = \begin{cases} x=0 \\ y+1=0 \\ z-1=0 \end{cases} \\
T = \begin{cases} y^2-1=0 \\ z-1=0 \end{cases} \arrow[ur, "y+1=0"] \arrow[dr, "y+1\neq 0"'] & & \\
 & T_2 = \begin{cases} y-1=0 \\ z-1=0 \end{cases} \arrow[r, "f = 2x^2 - x"] & T_4 = \begin{cases} 2x^2-x=0 \\ y-1=0 \\ z-1=0 \end{cases}
\end{tikzcd}$$

This actually forms a **direct product** isomorphism:

$$\mathbb{K}[x,y,z]/\mathrm{sat}(T) \;\cong\; \mathbb{K}[x,y,z]/\mathrm{sat}(T_1) \otimes \mathbb{K}[x,y,z]/\mathrm{sat}(T_2)$$

Quasi-Components and Triangular Decomposition

Quasi-component of a regular chain: Let $h_T = \prod_{p \in T} \mathrm{init}(p)$

- $W(T) := V(T) \setminus V(h_T)$
- $\overline{W(T)} = \overline{V(T) \setminus V(h_T)}$
- $W(\varnothing) = \overline{\mathbb{K}}^n$

A **triangular decomposition** of an input system $F \subseteq \mathbb{K}[\underline{X}]$ is a set of regular chains $T_1, \ldots, T_e$ such that:

(*Lazard-Wu decomposition*) $\quad V(F) = \bigcup_{i=1}^{e} W(T_i)$, or

(*Kalkbrener decomposition*) $\quad V(F) = \bigcup_{i=1}^{e} \overline{W(T_i)}$

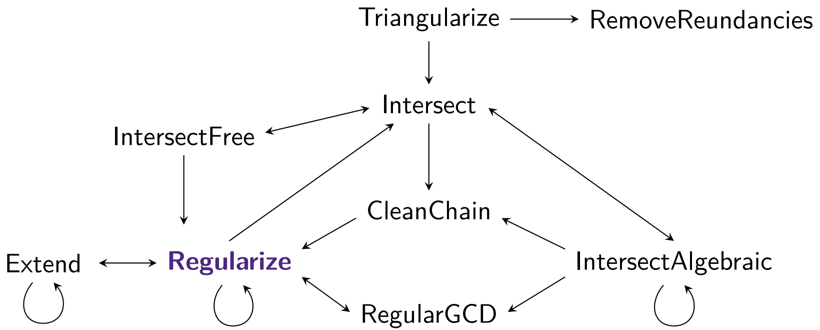
Some T_i may be **redundant**; $\exists j \; W(T_i) \subseteq W(T_j)$

- Should not return excessive solutions to client code/users
- Suggests some branches of computation are wasteful and unnecessary

All roads lead to Regularize

The Triangularize algorithm iteratively calls intersect, then a network of mutually recursive functions do the heavy-lifting.

↳ In all cases, polynomials are forced to be regular and splittings are (possibly) found via **Regularize**



Outline

Concurrency Opportunities

Component-level parallelism

- Concurrency in incremental decomposition: "triangularize tasks"
  ↳ **Map** (parallel for-loop), **Workpile** (queue with parallel while-loop)
- Concurrency between the many subroutines which call **Regularize**
  ↳ **Asynchronous Generators** (Producer-Consumer), **Pipeline**
- Removing redundant components
  ↳ Divide-and-Conquer like *mergesort* $\Longrightarrow$ **Fork-Join**

Low-level parallelism

- Subresultant chains
  ↳ Applies **Map** to computing modular images for interpolation and Chinese Remainder Theorem.
  ↳ Limited to univariate and bivariate subresultants
- Factorization, polynomial arithmetic (work in progress)

Triangularize: a task-based approach

Algorithm 1 TriangularizeByTasks(F)

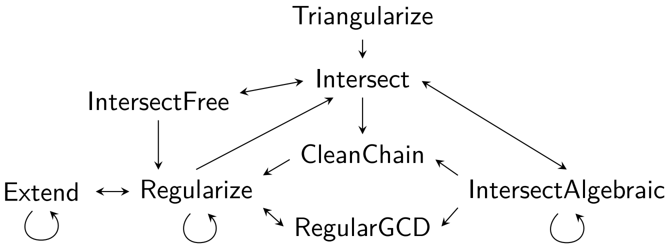
Input: a finite set $F \subseteq \mathbb{K}[\underline{X}]$

Output: regular chains $T_1, \ldots, T_e \subseteq \mathbb{K}[\underline{X}]$ such that $V(F) = W(T_1) \cup \cdots \cup W(T_e)$

Tasks $:= \{\ (F, \varnothing)\ \}$; $\mathcal{T} := \varnothing$
while $|$*Tasks*$| > 0$ **do**
    (P, T) := pop a task from *Tasks*
    Choose a polynomial $p \in P$; $P' := P \setminus \{p\}$
    for T' in **Intersect**(p, T) **do**
        if $|P'| = 0$ **then** $\mathcal{T} := \mathcal{T} \cup \{T'\}$
        else *Tasks* := *Tasks* $\cup \{(P', T')\}$
return RemoveRedundantComponents($\mathcal{T}$)

- Performs a *depth-first search*
- *Tasks* is essentially a data structure for a **task scheduler**
- A task can create more tasks, workers pop *Tasks* until none remain.
- Adaptive to load-balancing, no inter-task synchronization

Triangularize Subroutine Pipeline



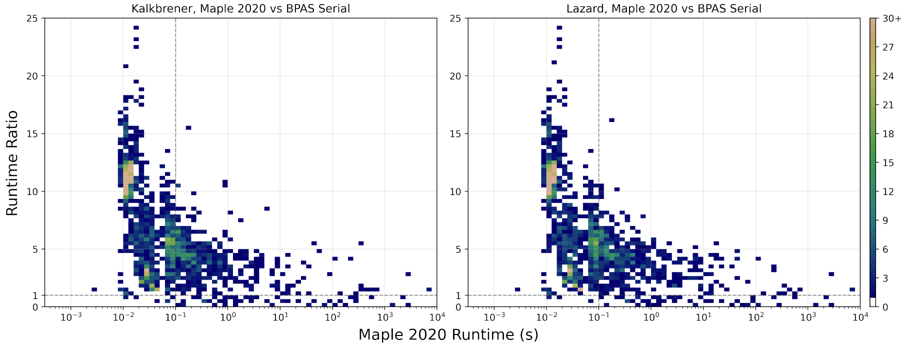
- Function call stack creates a **dynamic parallel pipeline** as several generators (producers) invoked and consumers process the data.
- Data **streams** between subroutines; all soubroutines are effectively *non-blocking*
- Pipeline creates **fine-grained parallelism** since work diminishes with each recursive call

Outline

Experimental Setup

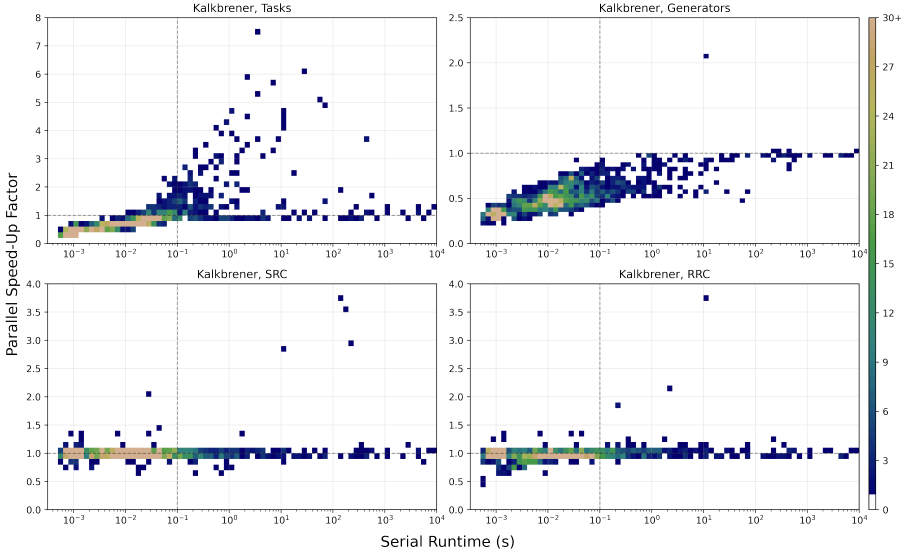
- A suite of >3000 polynomial systems has been compiled from systems in the **literature**, **user-data**, and **bug reports** provided by *Maplesoft*
- Only 1076 of these systems result in more than one component in their triangular decomposition
- In all other cases:
  - ↳ No speed-up expected from *component-level parallelism*
  - ↳ *Some* slow-down is expected, due to parallel overheads
- Four separate **parallel schemes** can be active or inactive
  - ↳ Triangualrize tasks, generators, removing redundancies, subresultants
- Experiments run on a node with two 6-core Intel Xeon X5650 CPUs
  - ↳ 24 physical threads with hyperthreading
  - ↳ 12x4GB DDR3 RAM at 1.33 GHz

Serial Performance

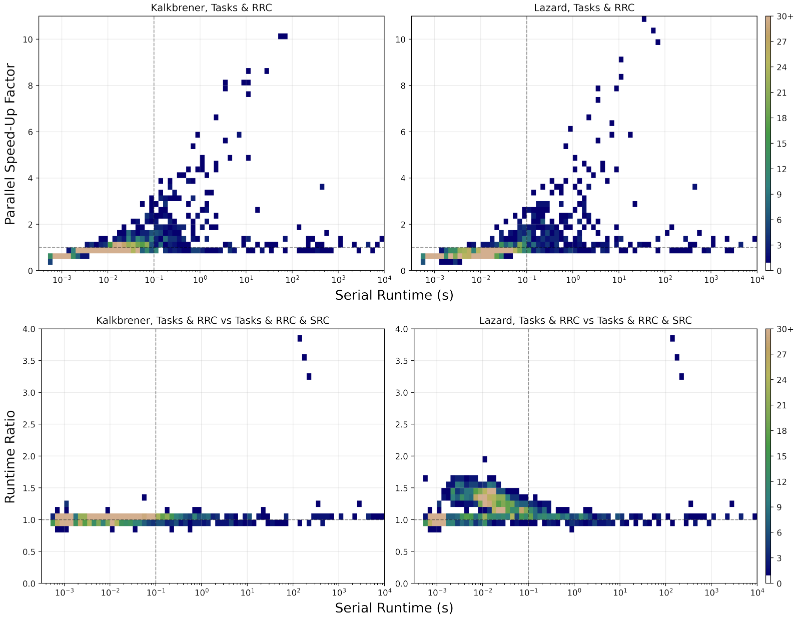


Serial triangular decomposition, BPAS vs *RegularChains* library of *Maple*

Performance of Individual Parallel Schemes



Performance of Combined Parallel Schemes



Outline

Avoiding Redundant Computations: Dynamic Evaluation

Two branches are likely to share geometric and algebraic features

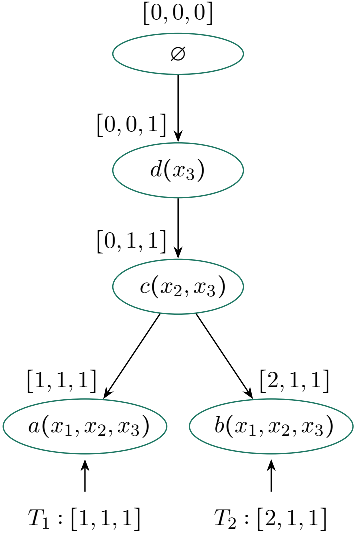
$$T_1 = \begin{cases} a(x,y) \\ c(y)d(y) \end{cases} \qquad T_2 = \begin{cases} b(x,y) \\ c(y)d(y) \end{cases}$$

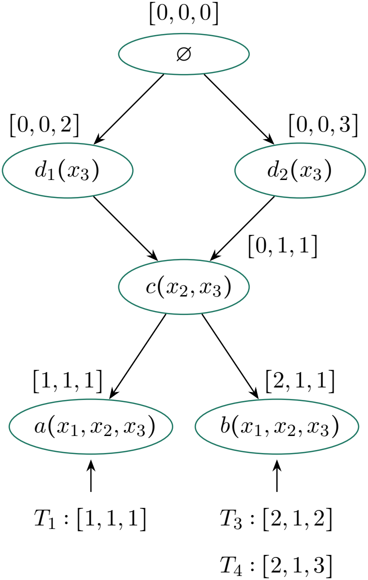
- Computations may split T_1 into $\{a(x,y),\ c(y)\}$ and $\{a(y,z),\ d(y)\}$
- T_2 hould automatically split into $\{b(x,y),\ c(y)\}$ and $\{d(y,z),\ d(y)\}$

Inspired by *cylindrical trees* in Cylindrical Algebraic Decomposition [10]

1. Each regular chain should exist only once in the universe
2. A split found in one regular chain should automatically be applied to other chains sharing that constraint
3. A unique and shared data structure $\Longrightarrow$ thread safety required

Regular Chains as Paths, Latent Splits





Outline

Triangular Decomposition, Limit Points

Triangular decomposition for an input set $F \subset \mathbb{K}[\underline{X}]$, find regular chains $T_1, \ldots, T_e$ such that:

- $V(F) = \overline{W(T_1)} \cup \overline{W(T_2)} \cup \cdots \cup \overline{W(T_e)}$ (Kalkbrener)
- $V(F) = W(T_1) \cup W(T_2) \cup \cdots \cup W(T_e)$ (Lazard-Wu)

In Kalkbrener decomp. $T_1, \ldots, T_e$ represent only **generic zeros** of $V(F)$

- Computing a Kalkbrener decomposition is *much* easier
- The **non-trivial limit points** of a regular chain are $\overline{W(T)} \setminus W(T)$.

Example:

$$T_1 = \begin{cases} bx + y \\ ay - b^2 \end{cases} \implies \begin{cases} x = \frac{-y}{b} \\ y = \frac{b^2}{a} \end{cases} \text{ where } b \neq 0,\ a \neq 0$$

$$\overline{W(T_1)} = W(T_1) \cup \begin{cases} x = 0 \\ y = 0 \\ b = 0 \end{cases} \cup \begin{cases} y = 0 \\ a = 0 \\ b = 0 \end{cases}$$

Computing Limit Points: Extended Hensel Construction

- Given a one-dimensional regular chain T, $\overline{W(T)}$ is an algebraic curve
- The limit points of $W(T)$ can be computed as *limits* of sequences of points along "branches" of an algebraic curve [1]
- Computing branches of an algebraic curve $F(X, Y)$ involves computing the roots of F in Y as Puiseux series in X

Newton-Puiseux Theorem:

$$F(X,Y) = (Y - f_1)\cdots(Y - f_d),\ f_i \text{ are Puiseux series in } X$$

Extended Hensel Construction (Hensel–Sasaki Construction):

$$F(X_1,\ldots,X_n,Y) = (Y - f_1)\cdots(Y - f_d),\ f_i \text{ are Puiseux series in } X_1,\ldots,X_n$$

↳ If F is monic, the f_i are **power series** in $X_1, \ldots, X_n$

Outline

Power Series: Definition

$\mathbb{A} = \mathbb{K}[[X_1, \ldots, X_n]]$ is the **ring of multivariate formal power series**

- Let $\mathbb{K}$ be algebraically closed.
- $f = \sum_e a_e X^e \in \mathbb{K}[[X_1, \ldots, X_n]]$
- $X^e = X_1^{e_1} \cdots X_n^{e_n}$, $|e| = e_1 + \cdots + e_n$
- **homogeneous part** of degree k: $f_{(k)} = \sum_{|e|=k} a_e X^e$
- $\mathcal{M} = \langle X_1, \ldots, X_n \rangle$ is the maximal ideal of $\mathbb{A} \Rightarrow f_{(k)} \in \mathcal{M}^k \setminus \mathcal{M}^{k+1}$

Example:

$f = 1 + X_1 + X_1X_2 + X_2^2 + X_1X_2^2 + X_1^3 + \cdots$ is known to **precision** 3

$$f_{(1)} = X_1 \qquad f_{(2)} = X_1X_2 + X_2^2 \qquad f_{(3)} = X_1X_2^2 + X_1^3$$

$\mathbb{A}[Y]$ is the ring of **Univariate Polynomials over Power Series** (UPoPS)

- $f = \sum_{i=0}^{d} a_i Y^i$, $a_i \in \mathbb{A}$, $a_d \neq 0$, is a UPoPS of degree d

Lazy Power Series: Design

Motivation: allow for terms to be computed on demand

1. Only compute terms explicitly needed:
   ↳ requested by user; needed for subsequent operations
2. Ability to **resume** and increase precision of an existing power series

Our lazy power series:

1. store previously computed homogeneous parts;
2. return previously computed homogeneous parts and, otherwise,
3. use an **update function** to compute homogeneous parts as needed;
4. capture parameters required for the update function.
   ↳ (3) and (4) effectively create a **closure**

Where update parameters are power series, they are called **ancestors**.

Addition, $f = g + h$

- $f_{(k)} = g_{(k)} + h_{(k)}$

Multiplication $f = gh$

- $f_{(k)} = \sum_{i=0}^{k} g_{(i)} h_{(k-i)}$

Ancestry Example

$$p = fg + ab$$

$$\begin{tikzcd}[column sep=small]
{f = \atop 1 + x + yz + \dots} \arrow[dr] & & {g = \atop 1 + z + y + \dots} \arrow[dl] & & {a = \atop 1 + y + x^2 + \dots} \arrow[dr] & & {b = \atop 1 + yz + xz + \dots} \arrow[dl] \\
& \times \arrow[d] & & & & \times \arrow[d] & \\
& {h = \atop 1 + z + y + x + yz + xz + xy + \dots} \arrow[drr] & & & & {c = \atop 1 + y + yz + xz + x^2 + \dots} \arrow[dll] & \\
& & & + \arrow[d] & & & \\
& & & {p = \atop 2 + z + 2y + x + 2yz + 2xz + xy + x^2 + \dots} & & &
\end{tikzcd}$$

Outline

Weierstrass Preparation: Informally

Weierstrass Preparation is a factorization of a UPoPS into two:
a *distinguished polynomial* and a unit

Let $f = a_{d+m}Y^{d+m} + a_dY^d + \cdots + a_2Y^2 + a_1Y + a_0$ be a UPoPS where:

- $a_{d+m}, \ldots, a_1, a_0 \in \mathbb{K}[[X_1, \ldots, X_n]]$
- $a_{d-1(0)} = \cdots = a_{0(0)} = 0$
- $m \in \mathbb{Z}_{\geq 0}$

Weierstrass Preparation Theorem tells us:

- $f = p\,\alpha$
- $p = Y^d + b_{d-1}Y^{d-1} + \cdots + b_1Y + b_0$, $b_{d-1(0)} = \cdots = b_{0(0)} = 0$
- α is an invertible element of $\mathbb{K}[[X_1, \ldots, X_n]][[Y]]$

A constructive proof of this theorem tells us that p and α can be computed lazily from power series arithmetic in $\mathcal{O}(dmk^2)$ operations in $\mathbb{K}$

Hensel Factorization

Algorithm 2 $\textsc{HenselFactorization}(f)$

Input: $f = Y^d + \sum_{i=0}^{d-1} a_i Y^i, a_i \in \mathbb{K}[[X_1, \ldots, X_n]]$.
Output: $f_1, \ldots, f_r$ s.t. $\prod_{i=1}^{r} f_i = f$, $f_i(0, \ldots, 0, Y) = (Y - c_i)^d_i$

$\bar{f} = f(0, \ldots, 0, Y)$
$(c_1, \ldots, c_r), (d_1, \ldots, d_r) :=$ roots and their multiplicities of $\bar{f}$
$\hat{f}_1 := f$
for $i := 1$ to $r - 1$ **do**
    $g_i := \hat{f}_i(Y + c_i)$
    $p_i, \alpha_i := \textsc{WeierstrassPreparation}(g_i)$
    $f_i := p_i(Y - c_i)$
    $\hat{f}_{i+1} := \alpha_i(Y - c_i)$
$f_r := \hat{f}_r$
return $f_1, \ldots, f_r$

Parallel Opportunities in Hensel

$$
\begin{tikzcd}
 & & p_1 \arrow[r, "-c_1"] & f_1 & & p_2 \arrow[r, "-c_2"] & f_2 & & p_3 \arrow[r, "-c_3"] & f_3 \\
f \arrow[r, "+c_1"] & g_1 \arrow[ur] \arrow[r] & \alpha_1 \arrow[r, "-c_1"] & \hat{f}_2 \arrow[r, "+c_2"] & g_2 \arrow[ur] \arrow[r] & \alpha_2 \arrow[r, "-c_2"] & \hat{f}_3 \arrow[r, "+c_3"] & g_3 \arrow[ur] \arrow[r] & \alpha_3 \arrow[r, "-c_3"] & f_4
\end{tikzcd}
$$

- The output of one Weierstrass becomes input to another
- $f_{i+i(k)}$ relies on $f_{i(k)}$
- Can compute $f_{i(k+1)}$ and $f_{i+i(k)}$ concurrently in a **pipeline**

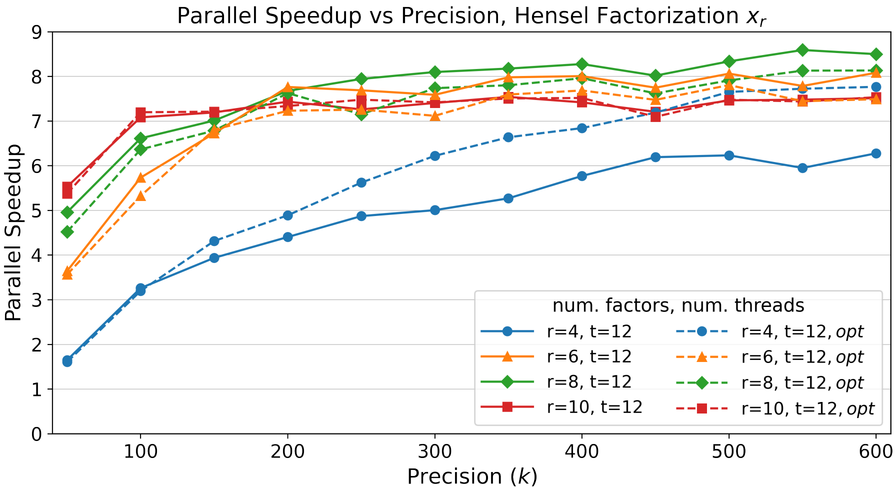
	Stage 1 (f_1)	Stage 2 (f_2)	Stage 3 (f_3)	Stage 4 (f_4)
Time 1	$f_{1(1)}$			
Time 2	$f_{1(2)}$	$f_{2(1)}$		
Time 3	$f_{1(3)}$	$f_{2(2)}$	$f_{3(1)}$	
Time 4	$f_{1(4)}$	$f_{2(3)}$	$f_{3(2)}$	$f_{4(1)}$
Time 5	$f_{1(5)}$	$f_{2(4)}$	$f_{3(3)}$	$f_{4(2)}$
Time 6	$f_{1(6)}$	$f_{2(5)}$	$f_{3(4)}$	$f_{4(3)}$

Parallel Challenges and Composition

$$\begin{tikzcd}
& & p_1 \arrow[r, "-c_1"] & f_1 & & p_2 \arrow[r, "-c_2"] & f_2 & & p_3 \arrow[r, "-c_3"] & f_3 \\
f \arrow[r, "+c_1"] & g_1 \arrow[ur] \arrow[r] & \alpha_1 \arrow[r, "-c_1"] & \hat{f}_2 \arrow[r, "+c_2"] & g_2 \arrow[ur] \arrow[r] & \alpha_2 \arrow[r, "-c_2"] & \hat{f}_3 \arrow[r, "+c_3"] & g_3 \arrow[ur] \arrow[r] & \alpha_3 \arrow[r, "-c_3"] & f_4
\end{tikzcd}$$

- Degrees and computational work diminish with each stage
  - ↳ $\deg(g_1) = d, \ \deg(g_2) = d - \deg(f_1), \ldots$
- Dominant cost to update f_i is WPT: $\mathcal{O}(\deg(p_i)\deg(\alpha_i)k^2)$
- To load-balance, execute WPT within each stage in parallel
- Assign t_i threads to stage i so that $\deg(p_i)\deg(\alpha_i)\,/\,t_i$ is equal for each stage.
- Better still, update a *group of successive factors* per stage.
  - ↳ To each stage s assign factors $f_{s_1}, \ldots, f_{s_2}$ and t_s threads so that $\sum_{i=s_1}^{s_2} \deg(p_i)\deg(\alpha_i)/t_s$ is roughly equal for each stage.

Parallel Speed-up Hensel Factorization



$$x_r = \prod_{i=1}^{r}(Y - i) + X_1(Y^3 + Y)$$

Outline

Conclusion

Our Contributions:

1. Algebraic class hierarchy
   - ↳ Compile-time mathematical type safety
   - ↳ “Make it hard to do the wrong thing”: ease-of-use, extensibility
2. Object-oriented, composable parallel framework
3. High-performance triangular decomposition
   - ↳ Speculative computation
   - ↳ Component-level parallelism
4. Algorithms and data structures to avoid redundant computation
5. Lazy & Parallel Hensel Factorization
   - ↳ Complexity estimates guide dynamic load-balancing

Future Work (1/2)

Parallel Computing & Software Design

- Further support for irregular parallelism
- New and hybrid parallel patterns, composition of patterns
- Cooperation of parallel regions
  - ↳ Gang scheduling, Cooperative multitasking
  - ↳ Dynamic resource re-distribution
  - ↳ Min/Max number of threads per region
- Quantitative profiling of irregular parallelism
  - ↳ How much concurrency was found?
  - ↳ How much parallelism was exploited?
  - ↳ *Tuning* of run-time parameters

Future Work (2/2)

Computer Algebra & Symbolic Computation

- Avoiding redundant computation in triangular decomposition
- Regular chain universe
  - ↳ Dynamic evaluation, latent splits, splitting trees
  - ↳ Adding parallelism requires efficient shared data structures
- Extend lazy-evaluation to Laurent series, Puiseux series
- Parallel pipeline for Extended Hensel Construction
- Improved thread distribution in Hensel pipeline: consider multivariate case and practical issues (coefficient sizes, locality)

References

[1] P. Alvandi, M. Ataei, M. Kazemi, and M. Moreno Maza. “On the Extended Hensel Construction and its application to the computation of real limit points”. In: *Journal of Symbolic Computation* 98 (2020), pp. 120–162.

[2] G. Attardi and C. Traverso. “Strategy-Accurate Parallel Buchberger Algorithms”. In: *Journal of Symbolic Computation* 22 (1996), pp. 1–15.

[3] P. Aubry, D. Lazard, and M. Moreno Maza. “On the Theories of Triangular Sets”. In: *Journal of Symbolic Computation* 28.1-2 (1999), pp. 105–124.

[4] J. Böhm, W. Decker, S. Laplagne, G. Pfister, A. Steenpaß, and S. Steidel. “Parallel algorithms for normalization”. In: *Journal of Symbolic Computation* 51 (2013), pp. 99–114.

[5] F. Boulier, F. Lemaire, and M. Moreno Maza. “Well known theorems on triangular systems and the D5 principle”. In: *Transgressive Computing 2006, Proceedings*. Granada, Spain, 2006.

[6] B. Buchberger. “The parallelization of critical-pair/completion procedures on the L-Machine”. In: *Japanese Symposium on Functional Pogramming, Proceedings*. 1987, pp. 54–61.

[7] C. Chen and M. Moreno Maza. “Algorithms for computing triangular decomposition of polynomial systems”. In: *Journal of Symbolic Computation* 47.6 (2012), pp. 610–642.

[8] C. Chen, J. H. Davenport, J. P. May, M. Moreno Maza, B. Xia, and R. Xiao. “Triangular decomposition of semi-algebraic systems”. In: *Journal of Symbolic Computation* 49 (2013), pp. 3–26.

[9] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. “Comprehensive Triangular Decomposition”. In: *Proc. of CASC 2007*. 2007, pp. 73–101.

[10] C. Chen and M. Moreno Maza. “An Incremental Algorithm for Computing Cylindrical Algebraic Decompositions”. In: *Proc. of Asian Symposium on Computer Mathematics ASCM 2012*. Springer, 2012, pp. 199–221.

[11] J. C. Faugere. "Parallelization of Gröbner Basis". In: *Proc. of PASCO 1994*. Vol. 5. World Scientific. 1994, p. 124.

[12] M. Gastineau and J. Laskar. "Parallel sparse multivariate polynomial division". In: *Proc. of PASCO 2015*. ACM, 2015, pp. 25–33.

[13] J. Grabmeier, E. Kaltofen, and V. Weispfenning, eds. *Computer algebra handbook*. Springer-Verlag, 2003.

[14] J. Hu and M. B. Monagan. "A Fast Parallel Sparse Polynomial GCD Algorithm". In: *Proc. of ISSAC 2016*. 2016, pp. 271–278.

[15] H. T. Kung and J. F. Traub. "All Algebraic Functions Can Be Computed Fast". In: *Journal of the ACM* 25.2 (1978), pp. 245–260.

[16] M. B. Monagan and R. Pearce. "Parallel sparse polynomial multiplication using heaps". In: *Proc. of ISSAC 2009*. ACM, 2009, pp. 263–270.

[17] M. B. Monagan and G. Paluck. "Linear Hensel Lifting for Zp[x, y] for n Factors with Cubic Cost". In: *ISSAC '22: International Symposium on Symbolic and Algebraic Computation, Villeneuve-d'Ascq, France, July 4 - 7, 2022*. Ed. by M. M. Maza and L. Zhi. ACM, 2022, pp. 159–166. DOI: 10.1145/3476446.3536178. URL: https://doi.org/10.1145/3476446.3536178.

[18] M. B. Monagan and B. Tuncer. "Sparse Multivariate Hensel Lifting: A High-Performance Design and Implementation". In: *Proc. of ICMS 2018*. 2018, pp. 359–368.

[19] M. Moreno Maza. *On Triangular Decompositions of Algebraic Varieties*. Tech. rep. TR 4/99. Presented at the MEGA-2000 Conference, Bath, England. Oxford, UK: NAG Ltd, 1999.

[20] M. Moreno Maza and Y. Xie. "Component-level parallelization of triangular decompositions". In: *Proc. of PASCO 2007*. ACM. 2007, pp. 69–77.

[21] B. D. Saunders, H. R. Lee, and S. K. Abdali. "A parallel implementation of the cylindrical algebraic decomposition algorithm". In: *Proc. of ISSAC 1989*. Vol. 89. 1989, pp. 298–307.

[22] W. Wu. "A zero structure theorem for polynomial equations solving". In: *MM Research Preprints* 1 (1987), pp. 2–12.

[23] W. Wu. "On zeros of algebra equations—an application of Ritt principle". In: *Kexeu Tongbao* 31.1 (1986), pp. 1–5.