Toward High-performance Polynomial System Solvers Based on Triangular Decompositions

Ph.D. Candidate: Xin Li
Supervisors: Marc Moreno Maza, Stephen M. Watt

Computer Science, University of Western Ontario
April 20, 2009

PART I: Introduction

- Motivations.
- Related Work.
- Goals and Contributions.

PART I / Motivations

- Solving polynomial systems is a driving subject for symbolic computation.
- Symbolic solvers require high computational power for large examples.
- Triangular decompositions are a highly promising technique for solving systems of algebraic equations symbolically.
- Asymptotically fast algorithms for polynomial arithmetic have been proven to be practically efficient.
- Designing and implementing high performance symbolic solvers based on triangular decompositions with the support of asymptotically fast algorithms is a very appealing subject.

PART I / Related Work (1/3)

- Gröbner bases and triangular decompositions are the main theoretical frameworks in symbolic polynomial system solving.
- Gröbner bases reveal the algebraic properties of the input system while triangular decomposition exhibits the geometry of its solution set.
- For a system with finitely many solutions, triangular decompositions can be computed within the same time bound as lexicographic Gröbner bases (Lazard 92) but space complexity seems to play in favor of triangular decompositions (Dahan 09).
- Regular chains (Kalkbrener 91) (Yang & Zhang 91)
  Polynomial GCDs modulo regular chains (Moreno Maza 2000)
  Equiprojectable decompositions (Dahan, Moreno Maza, Schost, Wu & Xie 05) are concepts which have contributed to improve methods for computing triangular decompositions.

PART I / Related Work (2/3)

- Let us consider the following system:

$$F : \begin{cases} f_1 = x^2 + y + z - 1 \\ f_2 = x + y^2 + z - 1 \\ f_3 = x + y + z^2 - 1 \end{cases} \quad \text{for } x > y > z.$$

- The lexicographical Gröbner basis of $\{f_1, f_2, f_3\}$ is:

$$\begin{cases} g_1 = x + y + z^2 - 1 \\ g_2 = y^2 - y - z^2 + z \\ g_3 = 2yz^2 + z^4 - z^2 \\ g_4 = -z^2 - 4z^4 + 4z^3 + z^6 \end{cases}$$

- A possible triangular decomposition of $\{f_1, f_2, f_3\}$ is:

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

PART I / Related Work (3/3)

- Fast arithmetic algorithms are well developed since the 60's for univariate polynomials over fields (Gathen & Gerhard 99)
- Software offering fast polynomial arithmetic
  - MAGMA, fast multivariate arithmetic (but not open source)
  - NTL, highly efficient FFT-based univariate arithmetic (but with some technical constraints).
- Other software which have inspired or supported our work:
  - SPIRAL, FFTW, automatically tuned code for numerical computations
  - the computer algebra system AXIOM,
  - the REGULARCHAINS library in MAPLE.

PART I / Contributions

- We have designed a set of improved algorithms: fast modular multiplication, fast bivariate solver, fast regular GCD, and fast regularity test.
- We have systemically investigated a set of implementation techniques adapted for asymptotically fast polynomial algorithms supporting triangular decompositions.
- We have realized a high performance software library in C language which implements all our reported new algorithms in this thesis. We made this library available for MAPLE system thus it can directly support MAPLE pre-existing higher level solvers in REGULARCHAINS.

PART I / Source of the following slides

- Part II Fast polynomial arithmetic *Implementation techniques for fast polynomial arithmetic in a high-level programming environment.* (A. Filatei, X. Li, M. Moreno Maza, É. Schost ISSAC 06)
- Part III Supporting Higher Level Algorithms in AXIOM *Efficient implementation of polynomial arithmetic in a multiple-level programming environment.* (X. Li, M. Moreno Maza ICMS 06).
- Part IV Operations Modulo a Triangular Set *Fast arithmetic for triangular sets: From theory to practice.* (X. Li, M. Moreno Maza, É. Schost ISSAC 97) *Multithreaded parallel implementation of arithmetic operations modulo a triangular set.* (X. Li, M. Moreno Maza PASCO-07)
- PART V Computations Modulo Regular Chains *Computations modulo regular chains.* (X. Li, M. Moreno Maza, W. Pan ISSAC'09)
- PART VI Software Library *The MODPN library: Bringing fast polynomial arithmetic into MAPLE.* (X. Li, M. Moreno Maza, R. Rasheed, É. Schost MICA'08)

PART II: Fast Polynomial Arithmetic

- What is fast polynomial arithmetic.
- Historical notes for the use of fast polynomial arithmetic.
- Implementation effort on fast polynomial arithmetic.
- Performance of the implementation.

PART II / Fast Polynomial Arithmetic

- **What is this?**
  Univariate polynomial multiplication, division, GCD, *etc.* with quasi-linear complexity, *i.e* $O(d \log d)$, where d is the degree bound of the input polynomials.
- **Multiplication time:**
  M(d) number of coefficient operations conducted for a univariate polynomial multiplication in degree less than d.

Classical Multiplication	$M(d) = 2d^2$
Karatsuba Multiplication	$M(d) = 9d^{1.59}$
FFT over an arbitrary ring	$M(d) = C\, d \log d \log \log d$

- **Example**: *Extended Euclidean Algorithm:*

EEA	$O(d^2)$
FEEA	$33M(d) \log d$

PART II / Fast Arithmetic: Historical Notes

- Asymptotically fast algorithms for polynomial and matrix arithmetic have been known for more than forty years, such as Karatsuba multiplication (1962), Cooley and Tukey FFT (1965), and Strassen (1969).
- Unfortunately, their impact on computer algebra systems has been reduced until recently.
- In the last decade, several software such as MAGMA, NTL, LINBOX for performing symbolic computations have put a great deal of effort on high performance based on asymptotically fast arithmetic.
- Why did we start our implementation from the scratch?

PART II / Implementation Effort

- Implementation levels:
  $\mathbf{F}_p \to \mathbf{F}_p[X] \to \mathbf{F}_p[X_1, \cdots, X_n] \to \mathbf{F}_p[X_1, \cdots, X_n]/\langle T \rangle$.
- $\mathbf{F}_p$: new integer reduction tricks.
- $\mathbf{F}_p[X]$: use of FFT/TFT, fast division, GCD, fast interpolation *etc.*, code optimization such as reducing cache misses, pipeline hazard, memory consumption, loop overhead also thread-level parallelism.
- $\mathbf{F}_p[X_1, \cdots, X_n]$: Extending the univariate arithmetic to multidimensional FFT/TFT, interpolation, subresultants *etc.*
- $\mathbf{F}_p[X_1, \cdots, X_n]/\langle T \rangle$: New algorithms such as multiplication modulo a monic triangular set,GCD and regularity test modulo regular chains (introduced in later slides).

PART II / Specialized Montgomery Reduction Trick

- Let $p = c2^n + 1$ be a prime for $c < 2^n$. E.g $p = 5{*}2^{15} + 1$.
- Let $\ell = \lceil \log_2(p) \rceil$ and let $R = 2^\ell$.
- Input: a and ω, both reduced modulo p,
- Output: A such that $A \equiv a\omega/R \bmod p$ and $-(p-1) < A < 2(p-1)$.

1. $M_1 = a\omega$
2. $(q_1, r_1) = (M_1 \text{ div } R, M_1 \bmod R)$
3. $M_2 = r_1 c 2^n$
4. $(q_2, r_2) = (M_2 \text{ div } R, M_2 \bmod R)$
5. $M_3 = r_2 c 2^n$
6. $q_3 = M_3 \text{ div } R$
7. $A = q_1 - q_2 + q_3$.

Proposition

We use 3 single precision multiplications. The original trick uses 2 single and 1 double precision.

PART II / Univariate Multiplication over $\mathbf{F}_p$

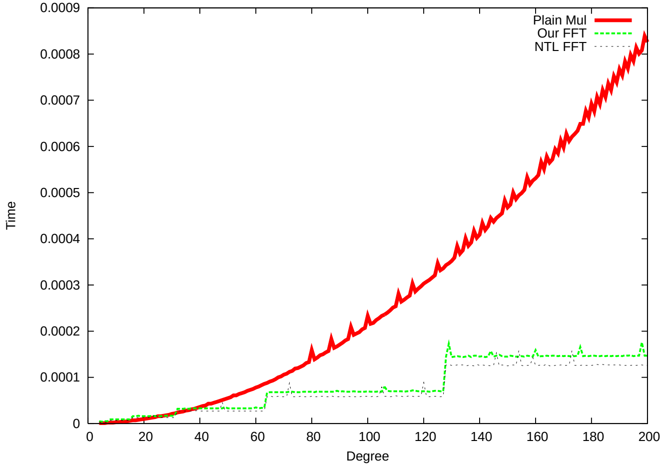


Figure: Univariate Multiplication Plain vs. Our FFT vs. NTL FFT.

PART II / Univariate Division over $\mathbf{F}_p$

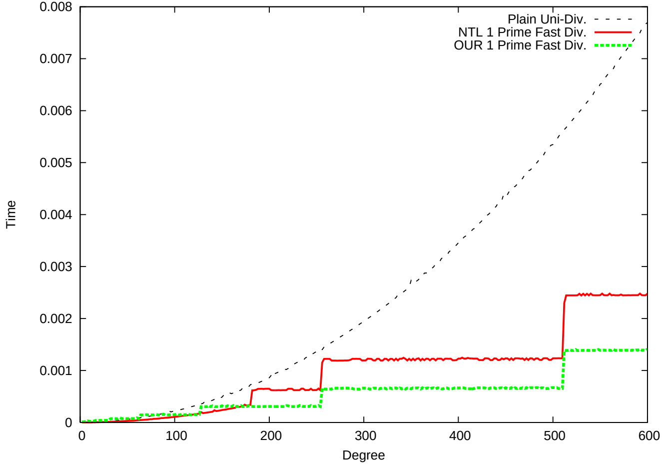


Figure: Univariate (Dense) Division Plain vs. Our Fast vs. NTL Fast

PART II / Truncated Fourier Transform

- Truncated Fourier Transform (TFT) (van der Hoeven 04).

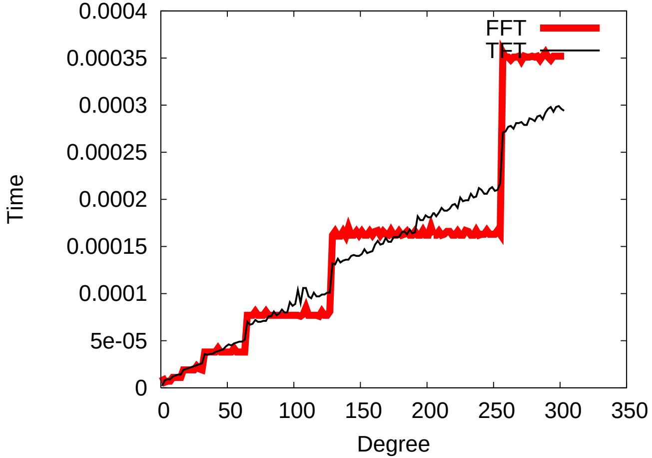


Figure: Univariate FFT vs. TFT

PART II / Multivariate Multiplication

- Compute the product of f_1 and f_2 in $\mathbb{Z}/p\mathbb{Z}[x_1, x_2, x_3]$, p is 64-bit precision prime number.

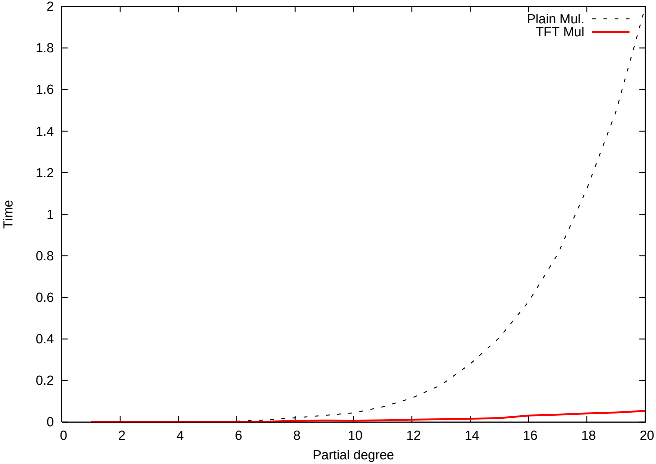


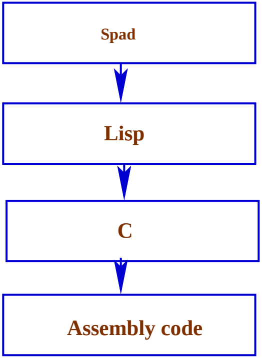
Figure: Plain vs. FFT

PART III: Supporting Higher Level Algorithms in AXIOM

- Why use AXIOM as the experimentation environment?
- Impact of the different data represenations.
- Impact of the programming languages.
- Can our fast arithmetic implementation speed up AXIOM high level pre-exsiting algorithms?

PART III / Speed Up Higher Level Pre-existing Algorithms in AXIOM

- Focus on implementation issues in AXIOM.
- Open AXIOM has a multiple-language level construction.

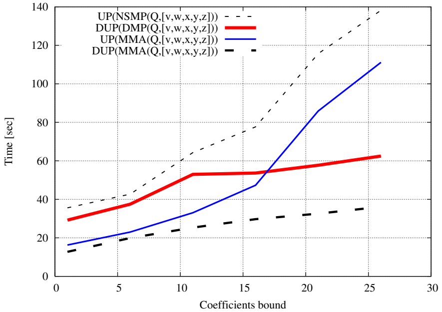


- Mixed code at each level for high performance.

PART III / Languages, types

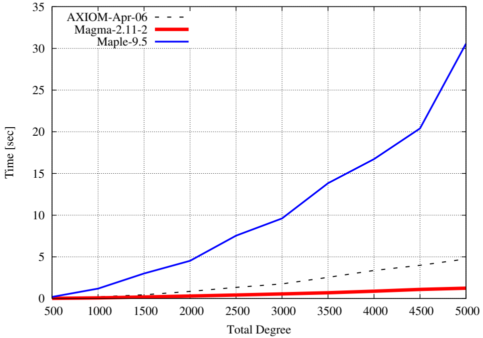
Benchmark: van Hoeij and Monagan Modular GCD algorithm
Input: $f_1, \; f_2 \in \mathbb{Q}(a_1, a_2, \ldots, a_e)[y]$
Output: $GCD(f_1, f_2)$

Multivariate Recursive	Univariate
SMP (sparse) in SPAD	SUP (sparse) in SPAD
DRMP (dense) in SPAD	DUP (dense) in SPAD
MMA (dense) in LISP	SUP (sparse) in SPAD
MMA (dense) in LISP	DUP (dense) in SPAD



PART III / Fast Arithmetics in C Supporting Higher Level algorithms

- Square-free factorization in $F_p[x]$.
- Comparing with MAPLE and MAGMA.



PART IV / Arithmetic Modulo Triangular Set

Monic Triangular Sets:
A family of polynomials $T = (T_1, \ldots, T_n)$ in $R[x_1 < \cdots < x_n]$, where R is a ring and each each $\mathrm{lc}(T_i, X_i) = 1$ and each T_i is reduced w.r.t. $\{T_1, \ldots, T_{i-1}\}$.

$$T \left| \begin{array}{l} T_n(x_1, \ldots, x_n) \\ \qquad \vdots \\ T_2(x_1, x_2) \\ T_1(x_1) \end{array} \right.$$

Fast arithmetic modulo monic triangular set :

- Well developed for univariate case. (Sieveking 72) (Kung 74)
- Still many challenges in multivariate case.

PART IV / Modular Multiplication

Example of modular multiplication.

$$\begin{array}{ll} Input: & P_1 = y^2 + x,\ P_2 = yx,\ and \quad T = \{x^2+1, y^3+x\}. \\ Ouput: & P_1 P_2\ mod\ T \ = \ 1 - y. \end{array}$$

Important observation of this operation.

- Modular multiplication is efficiency-critical to many other operations (*GCD, inversion, Hensel Lifting*), which are themselves the major sub-algorithm of polynomial system solvers based on triangular decomposition.
- Once optimized, it has the potential to bring huge speed-up factors to higher level operations.

PART IV / Theorem

Best Known complexity for modular multiplication:

- Input: A and B in $R[x_1, \ldots, x_n]$ reduced w.r.t. $\{T_1, \ldots, T_n\}$.
- Output: the product of $A\,B \mod \langle T_1, \ldots, T_n \}$.
- The size of input is $\delta_{\mathbf{T}} = \deg(T_1, x_1) \cdots \deg(T_n, x_n)$
- The total cost is $O\tilde{}(k^n \delta_{\mathbf{T}})$, k is a contant. ($O\tilde{}$ means we neglect log factors).
- The best known bound for $k \simeq 200$

Our improvement.

Theorem. Multiplications modulo $T_1(x_1), \ldots, T_n(x_1, \ldots, x_n)$ can be performed in $O\tilde{}(4^n \delta_{\mathbf{T}})$ base field operations.

ModMul($A, B, \{T_1, \ldots, T_n\}$)

1 $D := AB$ computed in $R[x_1, \ldots, x_n]$

2 **return** NormalForm$_n(D, \{T_1, \ldots, T_n\})$

PART IV / Algorithm

$\mathsf{NormalForm}_1(A : R[x_1], \{T_1 : R[x_1]\})$

$S_1 := \mathrm{Rev}(T_1)^{-1} \mod x_1^{\deg(A)-\deg(T_1)+1}$

$D := \mathrm{Rev}(A) S_1 \mod x_1^{\deg(A)-\deg(T_1)+1}$

$D := T_1 \, \mathrm{Rev}(D)$

return $A - D$

$\mathsf{NormalForm}_2(A : R[x_1, x_2], \{T_1 : R[x_1], T_2 : R[x_1, x_2]\})$

$A := \mathrm{map}(\mathsf{NormalForm}_1, \mathrm{Coeffs}(A, x_2), \{T_1\})$

$S_2 := \mathrm{Rev}(T_2)^{-1} \mod T_1, x_2^{\deg(A,x_2)-\deg(T_2,x_2)+1}$

$D := \mathrm{Rev}(A) S_2 \mod x_2^{\deg(A,x_2)-\deg(T_2,x_2)+1}$

$D := \mathrm{map}(\mathsf{NormalForm}_1, \mathrm{Coeffs}(D, x_2), \{T_1\})$

$D := T_2 \, \mathrm{Rev}(D)$

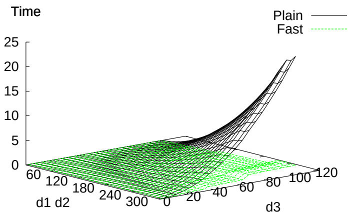
$D := \mathrm{map}(\mathsf{NormalForm}_1, \mathrm{Coeffs}(D, x_2), \{T_1\})$

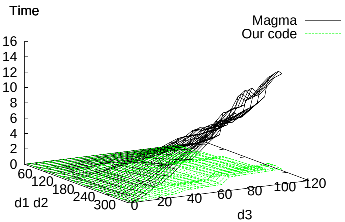
return $A - D$

PART IV / Performance

[left] comparison of classical (plain) and asymptotically fast strategies.

[right] comparison with MAGMA.



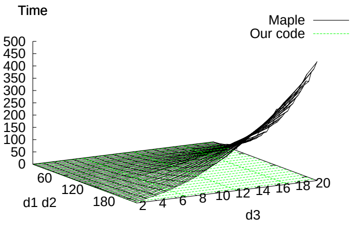


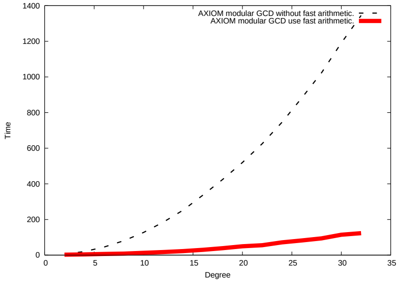
- Asymptotically fast strategy dominates the classical one.
- Our fast implementation is better than Magma's one (the best known implementation).

PART IV / Performance

[left] comparison with Maple's **recden** package, for GCD computations modulo a triangular set (over a finite field).

[right] comparison with AXIOM (our code vs. native arithmetic), for GCD computations in a number field.





- Huge factor comparing with the Maple's latest implementation.
- In AXIOM, replacing only the modular multivariate operation.

PART IV / Parallel bottom-up NormalForm

$\mathcal{P}$ = number of CPUs. s = number of variables.

<u>Thread-on-demand(f, TS, s)</u>

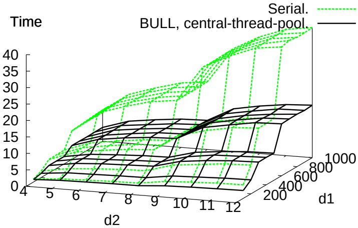
if $(s == 0)$ **return** f
$S \;=\; \prod_{j=1}^{s} (deg(f, x_j) + 1)$
$i = 1$
while $(i \leq s)$ **do**
  $ss = S \;/\; \prod_{j=1}^{i} (deg(f, x_j) + 1)$
    // suppose $\mathcal{P}$ divides ss.
  $q = ss \;/\; \mathcal{P}$
  for j **from** 0 **to** $\mathcal{P} - 1$ **repeat**
    $a = jp;\ b = (j+1)p$
    Task = **RS** (f, $a \cdots b$, TS, i)
    CreateThread (Task)
  $i = i + 1$
  DumpThreadPool()

<u>Central-thread-Pool(f, TS, s)</u>

Create $\mathcal{P}$ threads, sleep.
if $(s == 0)$ **return** f
$S \;=\; \prod_{j=1}^{s} (deg(f, x_j) + 1)$
$i = 1$
while $(i \leq s)$ **do**
  $ss = S \;/\; \prod_{j=1}^{i} (deg(f, x_j) + 1)$
    // suppose $\mathcal{P}$ divides ss.
  $q = ss \;/\; \mathcal{P}$
  for j **from** 0 **to** $\mathcal{P} - 1$ **repeat**
    $a = jp;\ b = (j+1)p$
    Task = **RS** (f, $a \cdots b$, TS, i)
    Wake up a thread to handle **Task**.
  $i = i + 1$
Finish and terminate all threads.

PART IV / Serial vs. Parallel NormalForm (central-thread-pool)

- AMD Opteron 4-processor 2.4 GHZ. Input $f \in F_p[x_1, x_2, x_3, x_4]$.



- With a small/medium number of relative large tasks, thread-on-demand and central-thread-pool have very similar performance.
- But, with a large number of relatively small tasks, the latter is slightly better in our application (shared memory).

PART V / Regular chain and GCD

- Let $T \subset \mathbf{k}[x_1 < \cdots < x_n] \setminus \mathbf{k}$ be a triangular set, hence the polynomials of T have pairwise distinct main variables.

- saturated ideal: $\mathrm{sat}(T) = \langle T \rangle : h_T^\infty$ with $h_T := \Pi_{C \in T} \mathrm{init}(C)$.
- T is regular chain if for each $C \in T$, with $v := \mathrm{mvar}(C)$, $\mathrm{init}(C) := \mathrm{lc}(C, v)$ is a regular modulo $\mathrm{sat}(T_{<v})$.

- Let $P, Q, G \in \mathbf{k}[x_1 < \cdots < x_n][y]$ be $\neq 0$ and T regular chain. G is a *regular GCD* of P, Q modulo $\mathrm{sat}(T)$ if
  - (i) $\mathrm{lc}(G, y)$ is a regular modulo $\mathrm{sat}(T)$,
  - (ii) $G \in \langle P, Q \rangle$ modulo $\mathrm{sat}(T)$,
  - (iii) $\deg_y(G) > 0 \;\Rightarrow\; \mathrm{prem}_y(P, G), \mathrm{prem}_y(Q, G) \in \mathrm{sat}(T)$.

- One can compute $T^1, \ldots, T^e$ and $G_1, \ldots, G_e$ such that G_i is a regular GCD of P, Q modulo $\mathrm{sat}(T_i)$ and

$$\sqrt{\mathrm{sat}(T)} = \cap_{i=0}^{e} \sqrt{\mathrm{sat}(T^i)}.$$

PART V / Main Result

- Let $P, Q \in \mathbf{k}[x_1 < \cdots < x_n][y]$ with $\mathrm{mvar}(P) = \mathrm{mvar}(Q) = y$.
- Let S_j for the j-th subresultant (w.r.t. y) of P, Q. Let $T \subset \mathbf{k}[x_1 < \cdots < x_n]$ be regular chain.
- Assume
  - $\mathrm{res}(P, Q, y) \in \mathrm{sat}(T)$,
  - $\mathrm{init}(P)$ and $\mathrm{init}(Q)$ are regular modulo $\mathrm{sat}(T)$,
  - Let $1 \leq d \leq \deg(Q, y)$ such that $S_j \in \mathrm{sat}(T)$ for all $0 \leq j < d$.
  - $\mathrm{lc}(S_d, y)$ is regular modulo $\mathrm{sat}(T)$,

Theorem

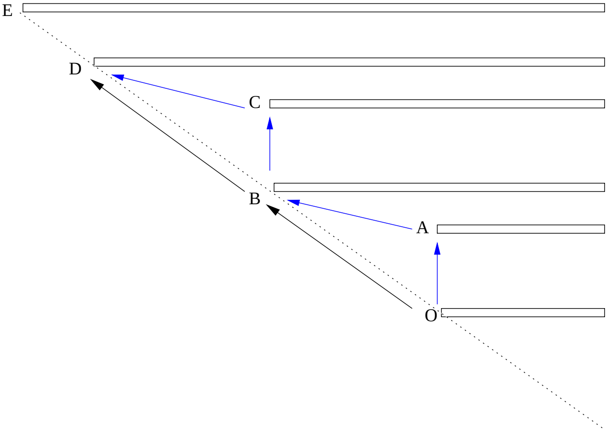
Assume that one of the following conditions holds:

- $\mathrm{sat}(T)$ *is radical,*
- *for all* $d < k \leq \mathrm{mdeg}(Q)$, *the coefficient of* y^k *in* S_k *is either null or regular modulo* $\mathrm{sat}(T)$.

Then, S_d *is a regular GCD of* P, Q *modulo* $\mathrm{sat}(T)$.

PART V / Algorithm

- Assume that the subresultants S_j for $1 \leq j < \mathrm{mdeg}(Q)$ are computed.
- Then one can compute a regular GCD of P, Q modulo $\mathrm{sat}(T)$ by performing a bottom-up search.



PART V / Complexity

- Let $x_{n+1} := y$. Define $d_i := \max(\deg(P, x_i), \deg(Q, x_i))$.
- Define $b_i := 2 d_i d_{n+1}$ and $B := (b_1 + 1) \cdots (b_n + 1)$.
- We compute S_j for $1 \le j < \mathrm{mdeg}(Q)$ by evaluation (via FFT) on an n-dimensional grid of points not cancelling $\mathrm{init}(P)$ and $\mathrm{init}(Q)$ in

$$O(d_{n+1} B \log(B) + d_{n+1}^2 B) \quad \text{where} \quad B \in O(2^n d_{n+1}^n d_1 \dots d_n).$$

- Then $\mathrm{res}(P, Q, y) = S_0$ is interpolated in time $O(B \log(B))$.
- When $\mathrm{sat}(T)$ is radical, neglecting the costs for regularity tests, a regular GCD is interpolated within $O(d_{n+1} B \log(B))$.
- If a regular GCD is expected to have degree 1 in y all computations fit in

$$O\tilde{}(d_{n+1} B).$$

PART V / Bivariate System Solving

- Let $P, Q \in \mathbf{k}[x_1 < x_2]$ with $\deg(P, x_2) \geq \deg(Q, x_2) > 0$. Assume $R := \mathrm{res}(P, Q, x_2) \notin \mathbf{k}$ and $\gcd(\mathrm{lc}(P, x_2), \mathrm{lc}(Q, x_2)) = 1$.
- Assume P, Q admits a regular GCD G modulo $\langle R \rangle$. Then we have
$$V(P, Q) = V(R, G).$$
- If $\deg(G, y) = 1$ then $V(P, Q)$ can be decomposed at the cost of computing R that is $O^{\sim}(d_2^2 d_1)$ operations in $\mathbf{k}$.
- Otherwise the decompsition is obtained within $O^{\sim}(d_2^3 d_1)$.

PART V / Regularity Test

- **Input:** T regular chain with $|T| = n$ and $Q \in \mathbf{k}[x_1 < \ldots < x_n]$.
- **Output:** yes if Q is regular or 0 w.r.t $\mathrm{sat}(T)$, a splitting of T otherwise.

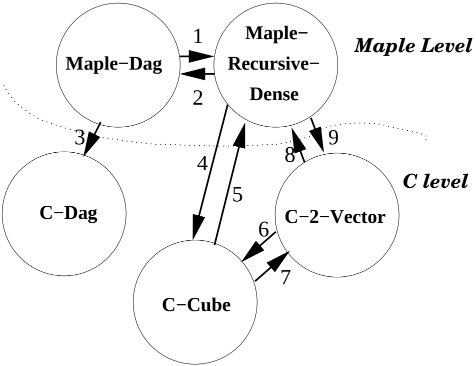
$Q := \mathsf{NormalForm}_n(Q, T)$
if $Q \in \mathbf{k}$ **then return** yes
$v := \mathrm{mvar}(Q)$
$R := \mathsf{res}(Q, T_v, v)$
if $R \equiv 0 \mod \mathrm{sat}(T_{<v})$ **then**
    $G := \mathsf{RegularGCD}(Q, T_v, T_{<v})$
    output $T_{<v} \cup G \cup T_{>v}$
    return $T_{<v} \cup \frac{T_v}{G} \cup T_{>v}$
if R regular mod $\mathrm{sat}(T_{<v})$ **then return** yes
if computations split **then** follow the branches

PART VI / The MODPN library (I)

MODPN is a MAPLE library implementing $\mathbb{Z}/p\mathbb{Z}[X_1, \ldots, X_n]$:

- highly efficient C implementations of key routines:
  - multivariate multiplication
  - normal form modulo a 0-dim regular chain
  - multivariate evaluation / interpolation
  - subresultant chain, iterated resultant
  - invertibility test modulo a 0-dim regular chain
- conversions **to** and **from** Maple representations (DAG / RECDEN).
- high-level algorithms written in MAPLE, supported by our C routines:
  - GCD of multivariate polynomials modulo a regular chain
  - regularity test of a polynomial modulo a regular chain
  - solver of a (square, 0-dim) polynomial systems

PART VI / The MODPN library (II)



- *C-Dag* for straight-line program.
- *C-Cube* for FFT-based computations.
- *C-2-Vector* for compact dense representation.
- *Maple-Dag* for calling REGULARCHAINS library.
- *Maple-Recursive-Dense* for calling RECDEN library.

PART VI / Benchmark

- Bivariate solver, random generic input systems.
- For the largest examples (having about 5700 solutions), the ratio is about 460/7.

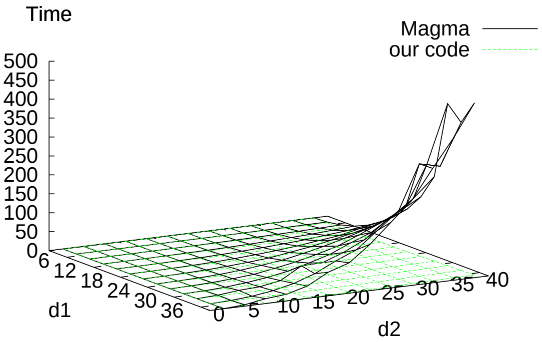


Figure: Generic bivariate systems: MAGMA vs. us.

PART VI / Benchmark

- Bivariate solver, designed examples to enforce many "splittings" (more branches of computations).
- For the largest examples, the ratio is about 5260/80, in our favor.

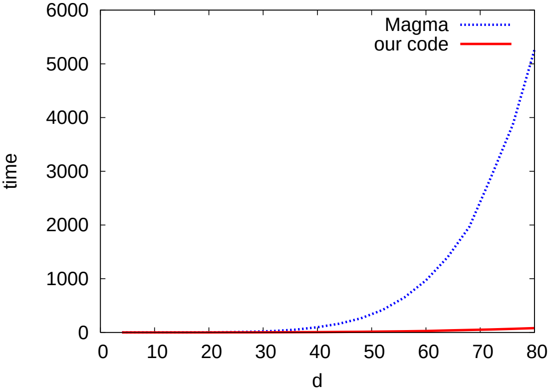


Figure: Non-generic bivariate systems: MAGMA vs. us.

PART VI / Benchmark

- Regularity test, brivariate input system, very few “splitting” (*possi*. = 2%).
- For the largest examples, the ratio is about 72/9, in our favor.

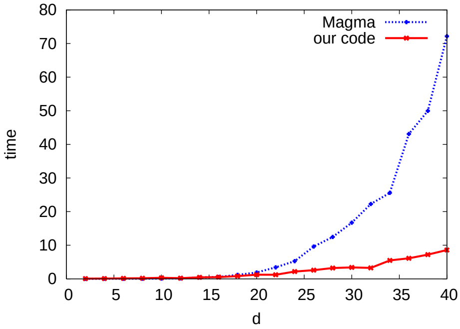


Figure: Bivariate case: timings, *possi*. = 2%.

PART VI / Benchmark

- Regularity test, brivariate input system, intensive "splitting" (*possi*. $= 50\%$).
- After partial degree 37, our code becomes faster.

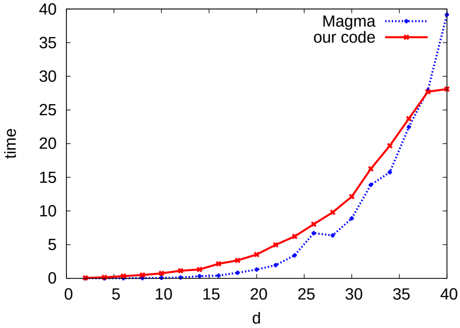


Figure: Bivariate case: timings, *possi*. $= 50\%$.

PART VI / Benchmark

Percentage MAPLE/C conversion time of the overall computation time:

- The profiling information for previous two benchmark examples.
- For *possi*. = 2% very few "splitting" case, reaches 60%.
- For *possi*. = 50% intensive "splitting" case, reaches 83%.

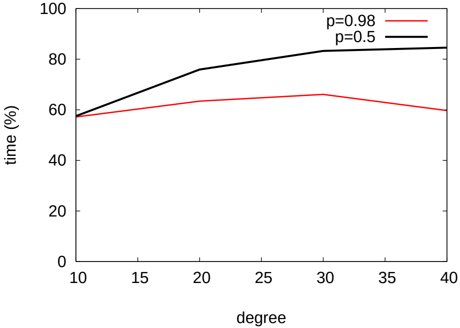


Figure: Bivariate case: time spent in conversions.

PART VI / Benchmark

- Regularity test.
- For relative small input system, the new fast code is already hundreds times faster than the pre-exiting code in MAPLE.

d_1	d_2	d_3	Regularize	Fast Regularize
2	2	3	0.292	0.012
3	4	6	1.732	0.028
4	6	9	68.972	0.072
5	8	12	328.296	0.204
6	10	15	>1000	0.652
7	12	18	>1000	2.284
8	14	21	>1000	5.108
9	16	24	>1000	18.501
10	18	27	>1000	31.349
11	20	30	>1000	55.931
12	22	33	>1000	101.642

Table: intensive "splitting" case 3-variable case.

Conclusion

- We have investigated and demonstrated that with suitable implementation techniques, FFT-based asymptotically fast polynomial arithmetic in practice can outperform the corresponding classical algorithms in a significant manner.
- By integrating our C-level implementation of fast polynomial arithmetic into AXIOM/MAPLE, the higher level pre-existing related libraries has been sped up in large scale.
- We have reported new algorithms, i.e. *modular multiplication*, *regular GCD*, and *regularity test*.
- In this research, we have focused on algorithms modulo regular chains in dimension-zero. Higher dimensional asymptotically fast triangular decompositions algorithms can be developed and implemented based on these results.

Reference

J. Grabmeier, E. Kaltofen, and V. Weispfenning, editors.
Computer Algebra Handbook.
Springer, 2003.

K. Geddes, S. Czapor, and G. Labahn.
Algorithms for Computer Algebra.
Kluwer Academic Publishers, 1992.

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.

R. T. Moenck.
Practical fast polynomial multiplication.
In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 136–148, New York, NY, USA, 1976. ACM Press.

D. H. Bailey, K. Lee, and H. D. Simon.
Using Strassen's algorithm to accelerate the solution of linear systems.
The Journal of Supercomputing, 4(4):357–371, 1990.

X. Li.
Efficient Management of Symbolic Computations with Polynomials.
2005.
University of Western Ontario

MAGMA: the computational algebra system for algebra, number theory and geometry.
http://magma.maths.usyd.edu.au/magma/.

NTL: the Number Theory Library.
http://www.shoup.net/ntl.

M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo.
SPIRAL: Code generation for DSP transforms.
Proc' IEEE, 93(2):232–275, 2005.

M. Frigo and S. G. Johnson.
Fftw.
http://www.fftw.org/.

F. Lemaire, M. Moreno Maza, and Y. Xie.
The `RegularChains` library.
In I. S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

A. Karatsuba and Y. Ofman.
Multiplication of multidigit numbers on automata.
Soviet Physics Doklady, (7):595–596, 1963.

J. Cooley and J. Tukey.
An algorithm for the machine calculation of complex Fourier +series.
Math. Comp., 19:297–301, 1965.

V. Strassen.
Gaussian elimination is not optimal.
Numerische Mathematik., 13:354–356, 1969.

J. Hoeven.
Truncated Fourier transform.
In *Proc. ISSAC'04*. ACM Press, 2004.

AXIOM: a general-purpose commercial computer algebra system.
http://page.axiom-developer.org.