Algorithmic Contributions to the Theory of Regular Chains

Wei PAN

January 25th, 2011

Solving a Polynomial System

Given a polynomial system in $\mathbb{Q}[x, y, z]$

$$\begin{cases} f_1 &= x^2 y - 2\,yz + 1, \\ f_2 &= xy^2 - z^2 + 2\,x, \\ f_3 &= y^2 z - x^2 + 5, \end{cases}$$

its lexicographic Gröbner basis with $x < y < z$ is

$$G = \begin{cases} -1 + 40800\,x + 60\,x^2 - 48320\,x^3 - 914\,x^4 + 20832\,x^5 + 2440\,x^6 - \\ 3840\,x^7 - 1652\,x^8 + 256\,x^9 + 400\,x^{10} + 25\,x^{11} - 32\,x^{12} - 10\,x^{13} + x^{15}, \\ \\ 14381563508743348530 + \cdots + 2227572578344303088 0\,x^3 + \\ 173379621293395768 00\,x^4 + \cdots + 6786531888173113 60\,x^7 + \\ 4178881948567437 05\,x^9 + \cdots + 1295293069959120\,x^{14} + \\ 1438088726195281921\,y, \\ \\ 733686935797091 20 + \cdots + 45195955594396940 16\,x^7 + \\ 44692331874910326 0\,x^8 + \cdots + 27806316011036620\,x^{12} + \\ 7748578966043840\,x^{13} - 5388716836938020\,x^{14} + \\ 1438088726195281921\,z \end{cases}$$

$\mathrm{length}(G) = 957.$

Solving a Polynomial System

Given a polynomial system in $\mathbb{Q}[x, y, z]$

$$\begin{cases} f_1 &= x^2y - 2\,yz + 1, \\ f_2 &= xy^2 - z^2 + 2\,x, \\ f_3 &= y^2z - x^2 + 5, \end{cases}$$

a triangular decomposition with $x < y < z$ is

$$T = \begin{cases} t_1 = & -1 + 40800\,x + 60\,x^2 - 48320\,x^3 - 914\,x^4 + 20832\,x^5 + 2440\,x^6 - \\ & 3840\,x^7 - 1652\,x^8 + 256\,x^9 + 400\,x^{10} + 25\,x^{11} - 32\,x^{12} - 10\,x^{13} + x^{15} \\ t_2 = & 40 - 408\,x^2 + 240\,x^4 + x^5 - 32\,x^6 - 10\,x^7 + 2\,x^9 + \\ & \left(x^7 - 80\,x^2 + 24\,x^4 + 4\right) y \\ t_3 = & 2\,yz - x^2y - 1. \end{cases}$$

$\mathrm{length}(T) = 286$.

Notations

$\mathtt{mvar}(t_1) = x$, $\mathtt{mvar}(t_2) = y$, $\mathtt{mvar}(t_3) = z$.

$\mathtt{init}(t_1) = 1$, $\mathtt{init}(t_2) = x^7 - 80\,x^2 + 24\,x^4 + 4$, $\mathtt{init}(t_3) = 2y$.

Key References

- Gröbner basis Bruno Buchberber in 1960's
- Characteristic set Joseph Fels Ritt in the 1930's and Wen Tsun Wu in the 1980's
- Regular chain Michael Kalkbrener and Normal ascending chain Lu Yang & Jingzhong Zhang in 1990's.
- Theory unification by Philippe Aubry, Daniel Lazard & Marc Moreno Maza in 1999.
- Triangular decomposition algorithms: (Dongming Wang 1993-1998-2000) (M. Kalkbrener 1991) (Daniel Lazard 1992) (M. Moreno Maza 2000) (Xavier Dahan, M. Moreno Maza, Éric Schost, Yuzhen Xie 2005) (Changbo Chen & M. Moreno Maza 2010).
- Software: The `RegularChains` library in MAPLE.

Contributions of this Thesis

- Better understanding of the theory of regular chains:
  - a notion of primitivity for regular chains and
  - efficient criterion to remove redundant components.
- Better algorithm for computing regular GCDs:
  - enhanced theoretical foundation of the regular GCDs
  - proposed a bottom-up algorithm which permits the use of fast polynomial arithmetic and modular methods.
- Parallelization by means of GPU computing:
  - efficient fast Fourier transform over finite fields
  - efficient subroutines for computing subresultants by values
  - compute resultants faster and solve bivariate systems faster

Contributions of this Thesis

- Better understanding of the theory of regular chains:
  - a notion of primitivity for regular chains and
  - efficient criterion to remove redundant components.
- Better algorithm for computing regular GCDs:
  - enhanced theoretical foundation of the regular GCDs
  - proposed a bottom-up algorithm which permits the use of fast polynomial arithmetic and modular methods.
- Parallelization by means of GPU computing:
  - efficient fast Fourier transform over finite fields
  - efficient subroutines for computing subresultants by values
  - compute resultants faster and solve bivariate systems faster

Contributions of this Thesis

- Better understanding of the theory of regular chains:
  - a notion of primitivity for regular chains and
  - efficient criterion to remove redundant components.
- Better algorithm for computing regular GCDs:
  - enhanced theoretical foundation of the regular GCDs
  - proposed a bottom-up algorithm which permits the use of fast polynomial arithmetic and modular methods.
- Parallelization by means of GPU computing:
  - efficient fast Fourier transform over finite fields
  - efficient subroutines for computing subresultants by values
  - compute resultants faster and solve bivariate systems faster

References for the Thesis

- *When does* $\langle T \rangle$ *equal* $\mathtt{sat}(T)$*?* (with François Lemaire, Marc Moreno Maza, and Yuzhen Xie), Journal of Symbolic Computation, accepted.
- *Computations modulo regular chains* (with Xin Li, Marc Moreno Maza), Proceedings of ISSAC 2009.
- *Fast polynomial multiplication on a GPU* (with Marc Moreno Maza), High Performance Computing Symposium 2010, Journal of Physics: Conference Series 256.
- *Solving bivariate polynomial systems on a GPU* (with Marc Moreno Maza), submitted.

Regularity

- Let $\mathbb{A}$ be a commutative ring.
- Element $x \in \mathbb{A}$ is regular iff $x\,y = 0$ implies $y = 0$.

Example

Consider $\mathbb{A} = \mathbb{Z}/12\mathbb{Z} = \{\bar{0}, \bar{1}, \ldots, \bar{11}\}$. Regular elements are

$$\{\bar{1}, \bar{5}, \bar{7}, \bar{11}\},$$

and zero-divisors are

$$\{\bar{2}, \bar{3}, \bar{4}, \bar{6}, \bar{8}, \bar{9}, \bar{10}\}.$$

Regular Chains and Saturated Ideals

Let $T = (t_1(x_1), \ldots, t_i(x_1, \ldots, x_i), \ldots) \subset \mathbf{k}[x_1 < \ldots < x_n]$ be a triangular set.

The **saturated ideal** $\mathtt{sat}(T)$ of a triangular set T is

$$\begin{aligned} \mathtt{sat}(T) &= \langle T \rangle : h^\infty \\ &= \{f \in \mathbf{k}[x_1, \ldots, x_n] \mid h^e f \in \langle T \rangle \text{ for some e} \geq 0\}, \end{aligned}$$

where h is the product of initials of t_i's.

Regular chain (recursive)

(1) if $T = \emptyset$, then it is a regular chain and $\mathtt{sat}(T) = \langle 0 \rangle$;

(2) if $T = C \cup \{p\}$, then T is a regular chain, iff C is a regular chain and $\mathtt{init}(p)$ is regular modulo $\mathtt{sat}(C)$.

An Example

- In $\mathbf{k}[x_1 < x_2 < x_3 < x_4]$, let

$$U = \{x_2\, x_3 + x_1\} \quad \text{and} \quad T = \{x_2\, x_3 + x_1, x_1\, x_4 + x_2\}.$$

We have

$$\mathtt{sat}(U) = \langle x_2\, x_3 + x_1 \rangle : x_2^\infty = \langle x_2\, x_3 + x_1 \rangle,$$

and x_1 is regular modulo $\mathtt{sat}(U)$. Hence T is a regular chain.

- The saturated ideal $\mathtt{sat}(T)$ may be strictly larger than $\langle T \rangle$.

$$\begin{aligned} \langle T \rangle &= \langle x_2\, x_3 + x_1, x_4\, x_3 - 1 \rangle \cap \langle x_1, x_2 \rangle, \\ \mathtt{sat}(T) &= \langle x_2\, x_3 + x_1, x_4\, x_3 - 1 \rangle. \end{aligned}$$

An Example

- In $\mathbf{k}[x_1 < x_2 < x_3 < x_4]$, let

$$U = \{x_2\, x_3 + x_1\} \quad \text{and} \quad T = \{x_2\, x_3 + x_1, x_1\, x_4 + x_2\}.$$

We have

$$\mathtt{sat}(U) = \langle x_2\, x_3 + x_1 \rangle : x_2^\infty = \langle x_2\, x_3 + x_1 \rangle,$$

and x_1 is regular modulo $\mathtt{sat}(U)$. Hence T is a regular chain.

- The saturated ideal $\mathtt{sat}(T)$ may be strictly larger than $\langle T \rangle$.

$$\begin{aligned} \langle T \rangle &= \langle x_2\, x_3 + x_1, x_4\, x_3 - 1 \rangle \cap \langle x_1, x_2 \rangle, \\ \mathtt{sat}(T) &= \langle x_2\, x_3 + x_1, x_4\, x_3 - 1 \rangle. \end{aligned}$$

Primitive Regular Chains

Inclusion Test for Regular Chains

- The **inclusion test** problem
  - Does $\mathtt{sat}(T) \subseteq \mathtt{sat}(U)$ hold?

  is hard to answer, which causes certain redundancy in the output of triangular decomposition algorithms in practice.

- If a system of generators of $\mathtt{sat}(T)$ is known, then the inclusion test reduces to the ideal membership problem.

- Our objectives are, in **positive** dimension,
  - characterizing the T's for which $\mathtt{sat}(T) = \langle T \rangle$ holds;
  - deciding $\mathtt{sat}(T) = \langle T \rangle$ without Gröbner basis computation.

Main Theorem

Primitivity

- Let $f \in \mathbb{A}[x]$ be a univariateof degree $d > 0$. Then

$$f = \mathtt{lc}(f)x^d + \mathtt{tail}(f)$$

is **weakly primitive** iff

  - $\mathtt{lc}(f)$ is invertible or,
  - $\mathtt{tail}(f)$ is regular modulo $\langle \mathtt{lc}(f) \rangle$.

- Let $T = C \cup \{p\}$ be a regular chain. Then T is **primitive** if
  - C is primitive and
  - p is a weakly primitive polynomial regarded as a univariate polynomial in its main variable over $\mathbf{k}[\mathbf{x}]/\langle C \rangle$.

Theorem

- Regular chain T is primitive iff $\langle T \rangle = \mathtt{sat}(T)$ holds.

Main Theorem

Primitivity

- Let $f \in \mathbb{A}[x]$ be a univariateof degree $d > 0$. Then

$$f = \mathtt{lc}(f)x^d + \mathtt{tail}(f)$$

  is **weakly primitive** iff
  - $\mathtt{lc}(f)$ is invertible or,
  - $\mathtt{tail}(f)$ is regular modulo $\langle \mathtt{lc}(f) \rangle$.

- Let $T = C \cup \{p\}$ be a regular chain. Then T is **primitive** if
  - C is primitive and
  - p is a weakly primitive polynomial regarded as a univariate polynomial in its main variable over $\mathbf{k}[\mathbf{x}]/\langle C \rangle$.

Theorem

- Regular chain T is primitive iff $\langle T \rangle = \mathtt{sat}(T)$ holds.

Main Theorem

Primitivity

- Let $f \in \mathbb{A}[x]$ be a univariateof degree $d > 0$. Then

$$f = \mathtt{lc}(f)x^d + \mathtt{tail}(f)$$

  is **weakly primitive** iff
  - $\mathtt{lc}(f)$ is invertible or,
  - $\mathtt{tail}(f)$ is regular modulo $\langle \mathtt{lc}(f) \rangle$.

- Let $T = C \cup \{p\}$ be a regular chain. Then T is **primitive** if
  - C is primitive and
  - p is a weakly primitive polynomial regarded as a univariate polynomial in its main variable over $\mathbf{k}[\mathbf{x}]/\langle C \rangle$.

Theorem

- Regular chain T is primitive iff $\langle T \rangle = \mathtt{sat}(T)$ holds.

Application to the Inclusion Test

- There is an efficient routine to check if a regular chain T is primitive. Implemented as a command **IsPrimitive** of the RegularChain library in Maple.

- The inclusion $\mathtt{sat}(T) \subseteq \mathtt{sat}(U)$ holds iff $\langle T \rangle \subseteq \mathtt{sat}(U)$ holds, provided that T is primitive. The latter can be efficiently checked by pseudo-division computations.

- This new criterion together with the previous criteria covers most cases in practice.

Regular GCD and a Bottom-up Algorithm

Subresultant and GCD

Let $x < y$ and

$$\begin{cases} F = 3y^4 + 6y^3 - yx^2 - 2x^2 + 2y + 4, \\ G = y^3x^2 - 2 - 2y^3 + x^2 \end{cases}$$

Subresultants

$$\begin{cases} S_2 &= (-x^6 + 3\,x^4 - 4)y + 6\,x^4 - 2\,x^6 - 8, \\ S_1 &= \left(x^{10} - 4\,x^8 + x^6 + 10\,x^4 - 4\,x^2 - 8\right) y + 2\,x^{10} \\ & \quad -8\,x^8 + 2\,x^6 + 20\,x^4 - 8\,x^2 - 16, \\ S_0 &= -7\,x^{14} + 35\,x^{12} - 21\,x^{10} - 119\,x^8 + 112\,x^6 \\ & \quad +168\,x^4 - 112\,x^2 - 112. \end{cases}$$

- S_0 has two squarefree factors $A_1 = x^2 + 1$ and $A_2 = x^2 - 2$.
- $\{(A_1, S_1), (A_2, F)\}$ is a triangular decomposition of F and G.

Regular GCD

- Let $P, Q \in \mathbb{A}[y]$ be non-constant with regular leading coefficients.

- G is a *regular GCD* of P, Q if we have:
  - $\mathrm{lc}(G, y)$ is regular in $\mathbb{A}$,
  - $G \in \langle P, Q \rangle$ in $\mathbb{A}[y]$,
  - $\deg(G, y) > 0 \quad \Rightarrow \quad \mathtt{prem}(P, G, y) = \mathtt{prem}(Q, G, y) = 0$.

- In practice $\mathbb{A} = \mathbf{k}[x_1, \ldots, x_n]/\mathrm{sat}(T)$ for a regular chain T.

- Such a regular GCD may not exist. However one can compute $\mathcal{I}_i = \mathtt{sat}(T_i)$ and non-zero polynomials G_i such that

$$\sqrt{\mathcal{I}} = \cap_{i=1}^{e} \sqrt{\mathcal{I}_i} \quad \text{and} \quad G_i \quad \text{regular GCD of} \quad P, Q \bmod \mathcal{I}_i$$

Regular GCD

- Let $P, Q \in \mathbb{A}[y]$ be non-constant with regular leading coefficients.

- G is a *regular GCD* of P, Q if we have:
  - $\mathrm{lc}(G, y)$ is regular in $\mathbb{A}$,
  - $G \in \langle P, Q \rangle$ in $\mathbb{A}[y]$,
  - $\deg(G, y) > 0 \quad \Rightarrow \quad \mathtt{prem}(P, G, y) = \mathtt{prem}(Q, G, y) = 0$.

- In practice $\mathbb{A} = \mathbf{k}[x_1, \ldots, x_n]/\mathrm{sat}(T)$ for a regular chain T.

- Such a regular GCD may not exist. However one can compute $\mathcal{I}_i = \mathtt{sat}(T_i)$ and non-zero polynomials G_i such that

$$\sqrt{\mathcal{I}} = \cap_{i=1}^{e} \sqrt{\mathcal{I}_i} \quad \text{and} \quad G_i \quad \text{regular GCD of} \quad P, Q \bmod \mathcal{I}_i$$

Related Work

- This regular GCD was proposed in (Moreno Maza 2000)
- In previous work (Kalkbrener 1993) and (Rioboo & Moreno Maza 1995), other regular GCDs modulo regular chains were introduced, but with limitations.
- In other work (Wang 2000), (Yang etc. 1995) and (Jean Della Dora, Claire Dicrescenzo, Dominique Duval 85), related techniques are used to construct triangular decompositions.
- Regular GCDs modulo regular chains generalize GCDs over towers of field extensions for which specialized algorithms are available, (van Hoeij and Monagan 2002 & 2004).
- The complexity of computing regular GCDs (Dahan, Moreno Maza, Schost, and Xie)

Main Theorem

Let $P, Q \in \mathbf{k}[x_1, \ldots, x_n][y]$ and $S_0, \ldots, S_{q-1}$ be their subresultants in y, and $T \subset \mathbf{k}[x_1, \ldots, x_n]$ be a regular chain.

S_d is called a candidate regular GCD of P, Q if the following hold

1. $S_i \in \mathtt{sat}(T)$ for all $0 \leq i < d$
2. $S_d \notin \mathtt{sat}(T)$
3. $\mathtt{init}(S_d)$ is regular modulo $\mathtt{sat}(T)$

Theorem

Candidate S_d is a regular GCD of P, Q w.r.t T, if for each $\ell > d$ either condition holds

1. $\mathtt{coeff}(S_\ell, y^\ell) \in \mathtt{sat}(T)$,
2. $\mathtt{coeff}(S_\ell, y^\ell)$ *is regular modulo* $\mathtt{sat}(T)$.

Implications of the Main Theorem

- Separate the computation of regular GCDs and that of subresultants
  - reduce the algebraic complexity by recycling subresultants
  - reduce the memory consumption

- Permit to use fast polynomial arithmetic by pushing computations to the ground field
  - represent subresultants by values (FFT evaluations)
  - interpolate subresultants only needed (FFT interpolations)
  - exposure the parallelism

Implications of the Main Theorem

- Separate the computation of regular GCDs and that of subresultants
  - reduce the algebraic complexity by recycling subresultants
  - reduce the memory consumption

- Permit to use fast polynomial arithmetic by pushing computations to the ground field
  - represent subresultants by values (FFT evaluations)
  - interpolate subresultants only needed (FFT interpolations)
  - exposure the parallelism

The Complexity of the Regular GCD Algorithm

- Let $d_i = \max(\deg(P, x_i), \deg(Q, x_i))$
- $d_{n+1} = \deg(Q, y)$ and $D_k = d_1 \cdots d_k$
- Assume that T zero-dimensional prime ideal

Corollary

If $d = d_1 = \cdots = d_{n+1} \geq 2$, *then*

$$\mathrm{GCD}(d_1, \ldots, d_n, d_{n+1}) \in O^{\sim}(n^2 2^n d^{2n+2}).$$

Theorem

$$\mathrm{GCD}(d_1, \ldots, d_n, d_{n+1}) \in O^{\sim}(n^2\, 2^n) d_{n+1}^{n+2} D_n + O^{\sim}(2^n) \sum_{i=2}^{n} \left(i^2\, d_i^i\, D_i\right).$$

Fast Fourier Transform on GPUs

Fast Fourier Transform on GPUs

Background

- FFTs over finite fields is the starting point for asymptotically fast algorithms in symbolic computation.
- In the literature, most FFTs on GPUs are for complex numbers, such as NVIDIA CUFFT library.

Main Results

- Our Stockham FFT implementation is highly efficient. Even though the ratio between the algebraic complexity and the output size is $\Theta(\log n)$, it still achieves a reasonable amount speedup.
- Our CUDA kernels can be extended to realize a list of 1D FFTs, one of the building blocks for computing subresultants.

Bandwidth for CPUs and GPUs

Testing in GB/s

$\log_2 n$	memset	Main Mem to GPU	GPU to Main Mem	GPU Kernel
23	1.56	1.33	1.52	61.6
24	1.56	1.34	1.52	69.9
25	1.39	1.35	1.53	75.0
26	1.39	1.28	1.50	77.4
27	1.43	1.35	1.49	79.0

- "memset" for CPU to main memory
- "cudaMemcpy" between main memory and GPU memory
- Intel Core 2 Quad Q9400 @ 2.66GHz, 6GB memory, memory interface width 128 bits
- GeForce GTX 285, 1GB global memory, 30×8 cores, memory interface width 512 bits

Discrete Fourier Transform (DFT)

Definition

Given a primitive n-th root of unity ω (i.e. $\omega^{n/2} = -1$), and

$$f(t) = x_0 + x_1 t + \cdots + x_{n-1} t^{n-1},$$

$\mathrm{DFT}_n^{\omega}(f)$ is $\mathbf{y} = (y_0, \ldots, y_{n-1})$ with $y_k = f(\omega^k)$ for $0 \leq k < n$. As a matrix-vector product, it is

$$\mathbf{y} = \mathrm{DFT}_n \mathbf{x}, \quad \mathrm{DFT}_n = [\omega^{k\ell}]_{0 \leq k, \ell < n}. \tag{1}$$

Example

$$\begin{cases} y_0 &= x_0 + x_1 \\ y_1 &= x_0 - x_1 \end{cases} \iff \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

That is, $\mathrm{DFT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

Kronecker Product

The Kronecker (or tensor) product of A and B is

$$A \otimes B = [a_{k\ell}B]_{k,\ell} \quad \text{with} \quad A = [a_{k\ell}]_{k,\ell}$$

For example, let

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{and} \quad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

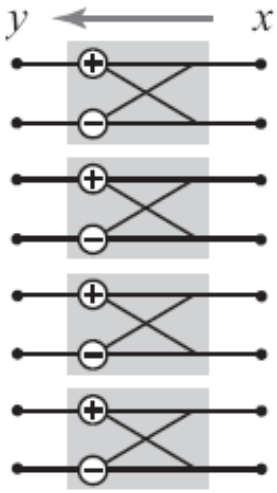
Then their tensor products are

$$I \otimes A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 3 & 4 \end{bmatrix} \quad \text{and} \quad A \otimes I = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 0 & 4 & 0 \\ 0 & 3 & 0 & 4 \end{bmatrix}.$$

Extract Parallelism from Structural Formulas

$I_n \otimes A$: block parallelism

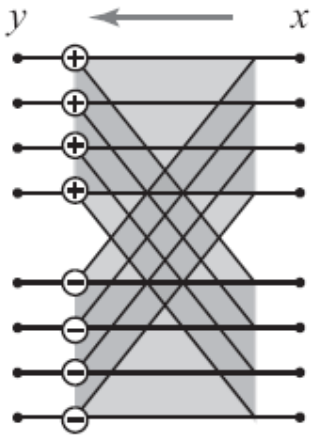
$$I_4 \otimes \mathrm{DFT}_2 = \begin{bmatrix} 1 & 1 & & & & & & \\ 1 & -1 & & & & & & \\ & & 1 & 1 & & & & \\ & & 1 & -1 & & & & \\ & & & & 1 & 1 & & \\ & & & & 1 & -1 & & \\ & & & & & & 1 & 1 \\ & & & & & & 1 & -1 \end{bmatrix}$$



Extract Parallelism from Structural Formulas

$A \otimes I_n$: vector parallelism

$$\mathrm{DFT}_2 \otimes I_4 = \begin{bmatrix} 1 & & & & 1 & & & \\ & 1 & & & & 1 & & \\ & & 1 & & & & 1 & \\ & & & 1 & & & & 1 \\ 1 & & & & -1 & & & \\ & 1 & & & & -1 & & \\ & & 1 & & & & -1 & \\ & & & 1 & & & & -1 \end{bmatrix}$$



Stockham FFT

$$\mathrm{DFT}_{2^k} = \prod_{i=0}^{k-1} \underbrace{(\mathrm{DFT}_2 \otimes I_{2^{k-1}})}_{\text{butterfly}} \underbrace{(D_{2,2^{k-i-1}} \otimes I_{2^i})}_{\text{twiddling}} \underbrace{(L_2^{2^{k-i}} \otimes I_{2^i})}_{\text{reordering}}$$

```
void stockham_dev(int *X_d, int n, int k, const int *W_d, int p)
{
  int *Y_d;
  cudaMalloc((void **)&Y_d, sizeof(int) * n);
  butterfly_dev(Y_d, X_d, k, p);
  for (int i = k - 2; i >= 0; --i) {
    stride_transpose2_dev(X_d, Y_d, k, i);
    stride_twiddle2_dev(X_d, W_d, k, i, p);
    butterfly_dev(Y_d, X_d, k, p);
  }
  cudaMemcpy(X_d, Y_d, sizeof(int)*n, cudaMemcpyDeviceToDevice);
  cudaFree(Y_d);
}
```

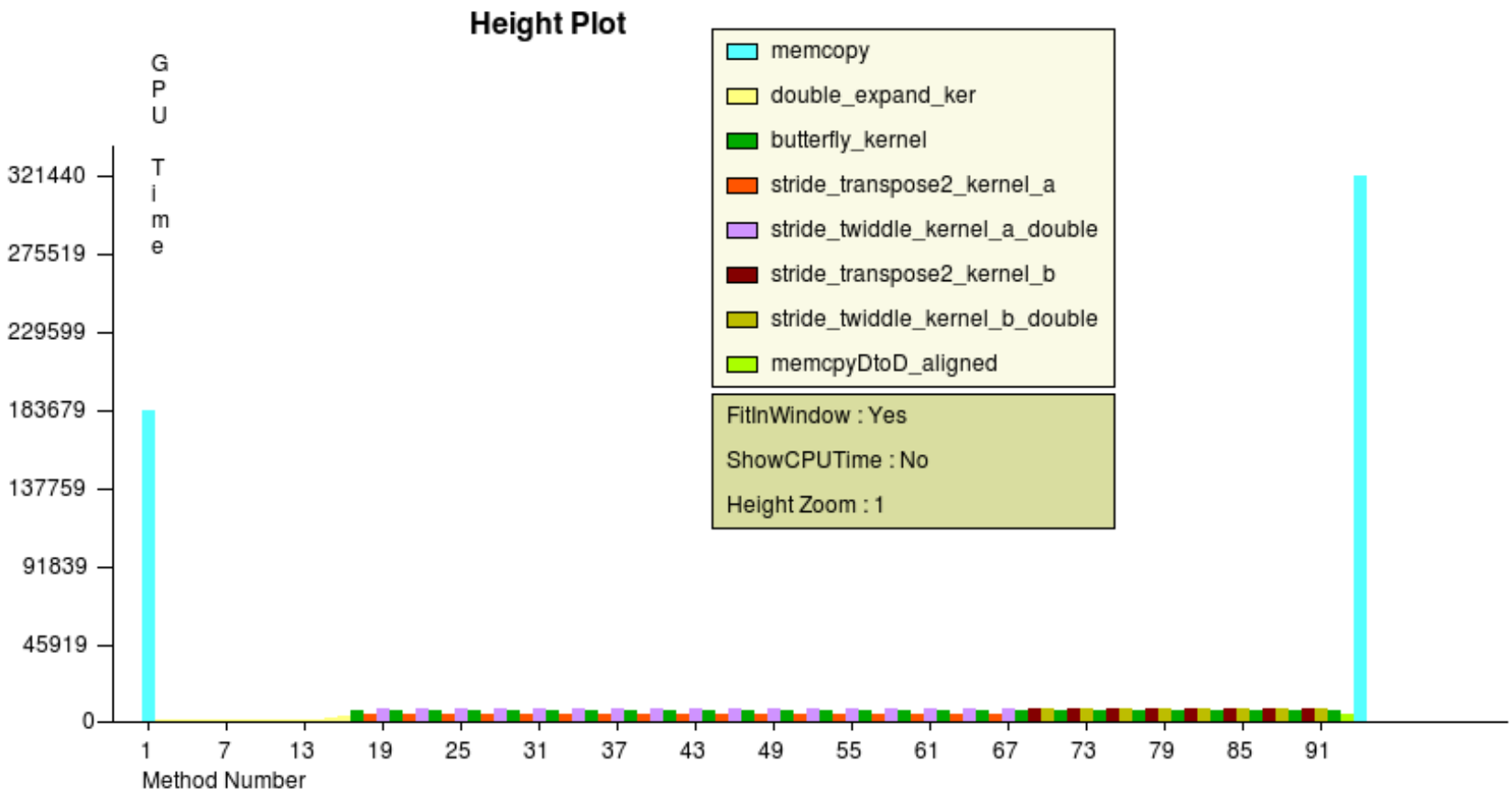
Stockham FFT

$$\mathrm{DFT}_{2^k} = \prod_{i=0}^{k-1} \underbrace{(\mathrm{DFT}_2 \otimes I_{2^{k-1}})}_{\text{butterfly}} \underbrace{(D_{2,2^{k-i-1}} \otimes I_{2^i})}_{\text{twiddling}} \underbrace{(L_2^{2^{k-i}} \otimes I_{2^i})}_{\text{reordering}}$$

```
__global__ void butterfly_ker(int *Y, const int *X, int k, int p)
{
  int bid = blockIdx.y * gridDim.x + blockIdx.x;
  int halfn = ((int)1 << (k - 1));
  const int *A = X + bid * blockDim.x;
  int *B = Y + bid * blockDim.x;
  int m = threadIdx.x + halfn;

  B[threadIdx.x] = add_mod(A[threadIdx.x], A[m], p);
  B[m] = sub_mod(A[threadIdx.x], A[m], p);
}
```

Implementation of Stockham FFT



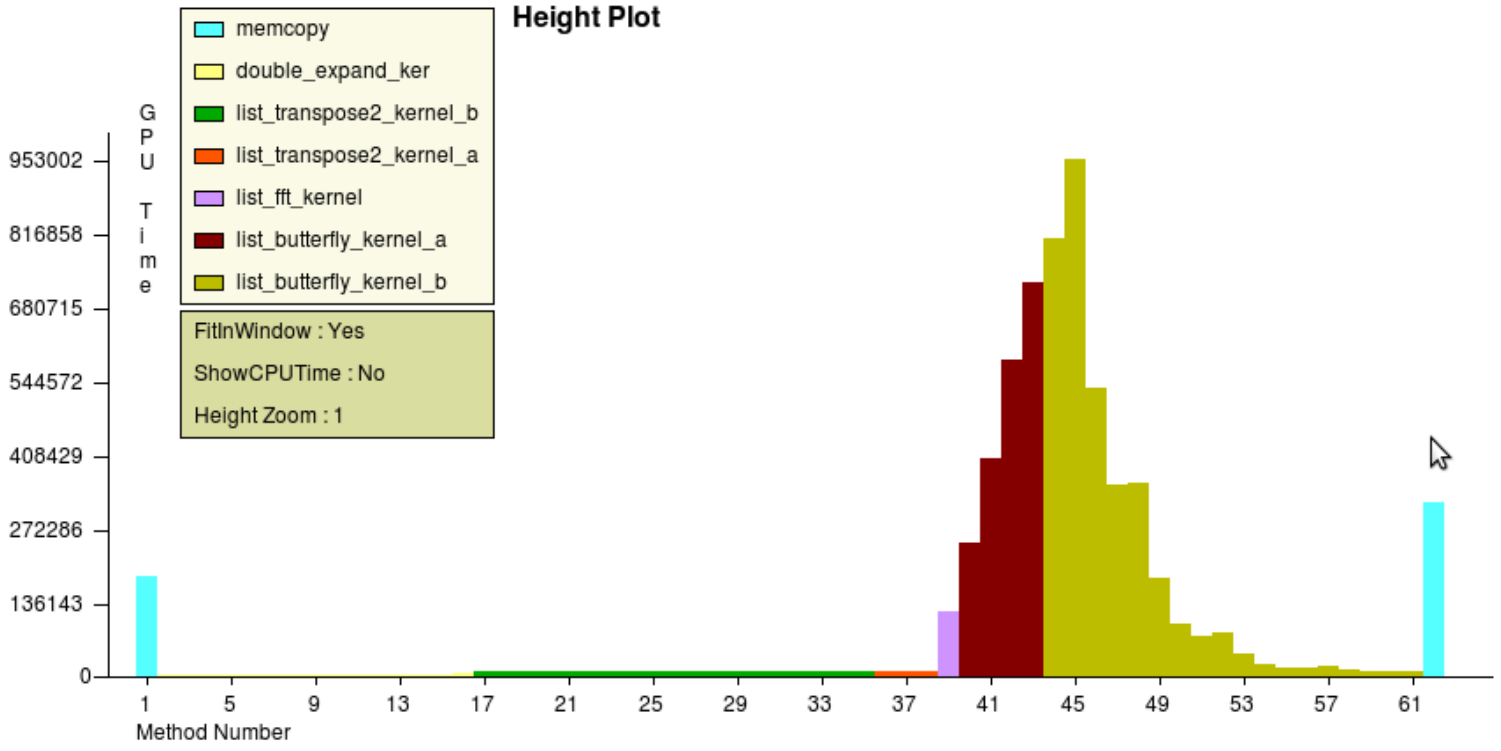
Cooley-Tukey FFT

$$\mathrm{DFT}_{2^k} = \left(\prod_{i=1}^{k} \left(I_{2^{i-1}} \otimes \mathrm{DFT}_2 \otimes I_{2^{k-i}} \right) T_{n,i} \right) R_n$$

with the twiddle factor matrix $T_{n,i} = I_{2^{i-1}} \otimes D_{2,2^{k-i}}$ and the bit-reversal permutation matrix

$$R_n = (I_{n/2} \otimes L_2^2)(I_{n/2^2} \otimes L_2^4) \cdots (I_1 \otimes L_2^n).$$

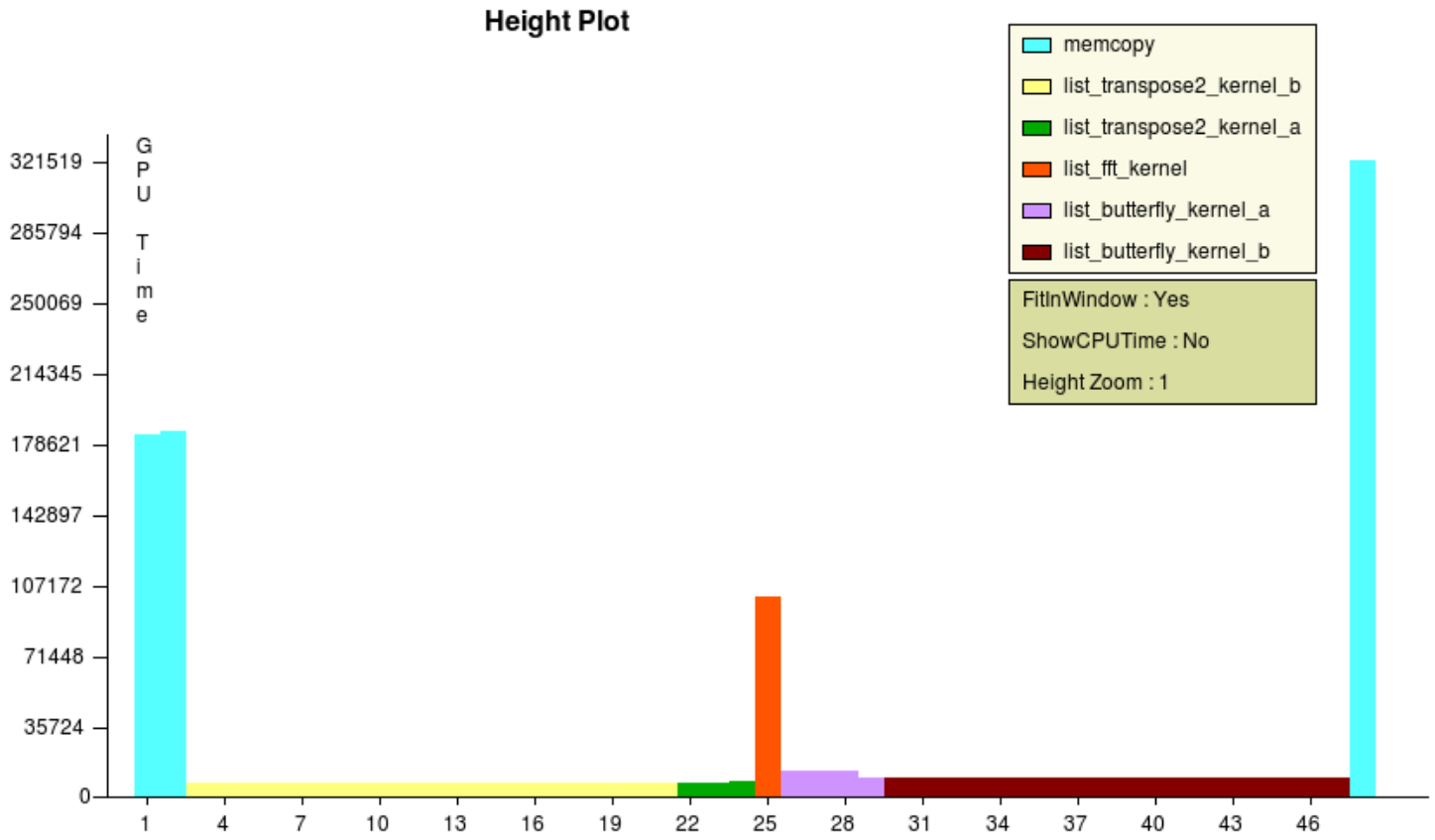
Implementation of Cooley-Tukey FFT



- The problem comes from the read to the "scattered" powers

$$1, \omega^{2^i}, (\omega^{2^i})^2, \ldots, (\omega^{2^i})^{2^{k-i}-1}$$

Implementation of Cooley-Tukey FFT



- Pre-compute "scattered" powers for each i

$$1, \omega^{2^i}, (\omega^{2^i})^2, \ldots, (\omega^{2^i})^{2^{k-i}-1}$$

Timing FFT in Milliseconds

e	modpn	Cooley-Tukey		C-T + Mem		Stockham		S + Mem	
		time	ratio	time	ratio	time	ratio	time	ratio
12	1	1	1.0	1	1.0	2	0.5	2	0.5
13	1	2	0.5	2	0.5	2	0.5	3	0.3
14	3	1	3.0	2	1.5	2	1.5	3	1.0
15	4	2	2.0	2	2.0	3	2.0	3	1.3
16	10	3	3.3	3	3.3	3	3.3	4	3.3
17	16	4	4.0	5	3.2	3	5.3	5	3.2
18	37	6	6.2	9	4.1	4	9.3	7	5.3
19	71	11	6.5	15	6.5	6	11.8	10	7.1
20	174	22	7.9	28	6.2	9	19.3	16	10.9
21	470	44	10.7	56	8.4	16	29.4	28	16.8
22	997	83	12.0	105	9.5	29	34.4	52	19.2
23	2070	165	12.5	210	9.9	56	37.0	101	20.5
24	4194	330	12.7	418	10.0	113	37.0	201	20.9
25	8611	667	12.9	842	10.2	230	37.4	405	21.2
26	17617	1338	13.2	1686	10.4	473	37.2	822	21.4

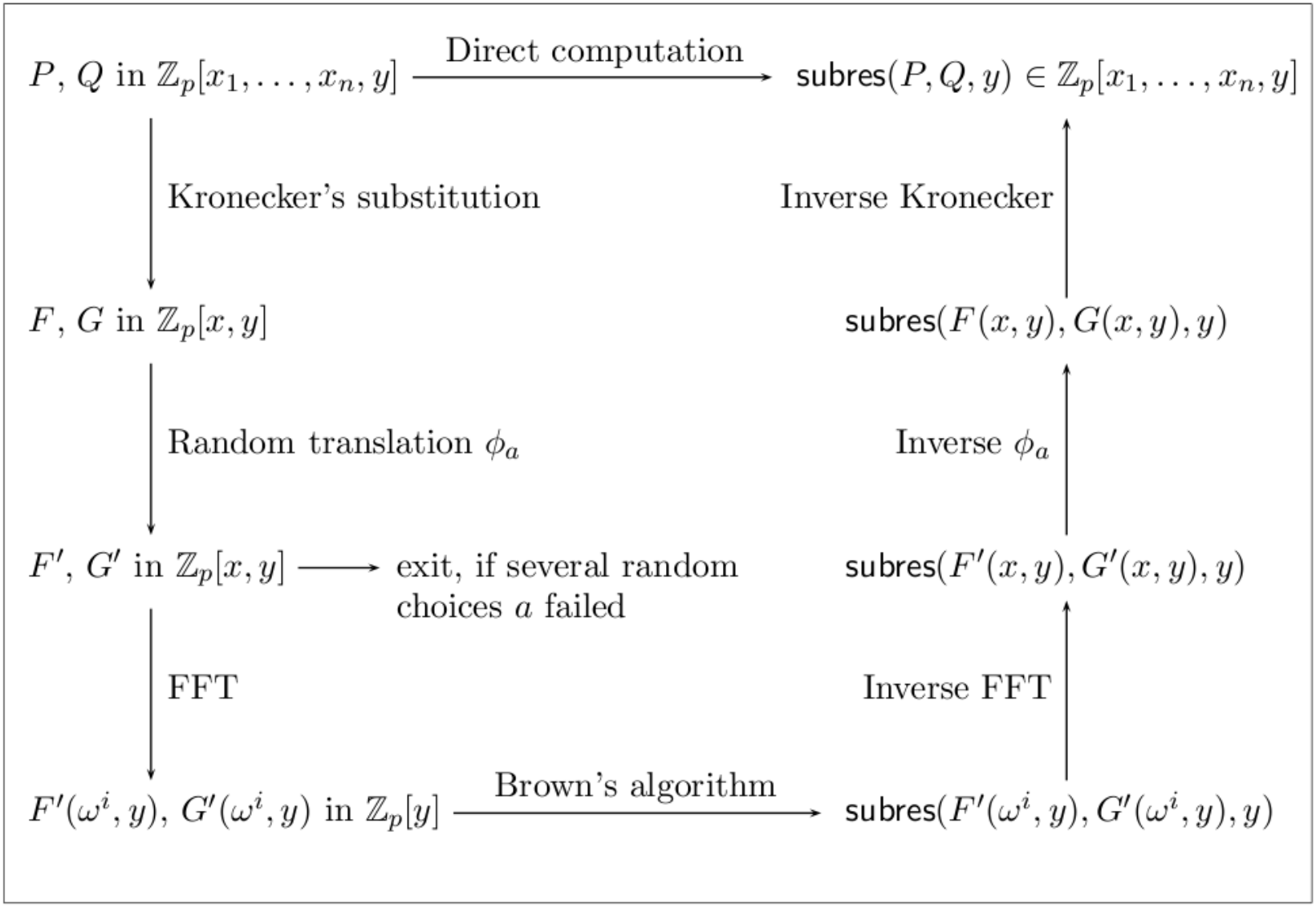
The GPU is GTX 285.

Summary

- The Stockham FFT achieves a speedup factor of 21 for large FFT degrees, comparing to the `modpn` serial implementation.
- The data transfer between GPU memory and main memory is approximately 42 %.

Computing Subresultants on the GPU

Subresultant Chain Computation



Space Complexity

Assume that

(a) $\max(\deg(P, x_i), \deg(Q, x_i)) \leq d_i$, for $1 \leq i \leq n$,

(b) $\deg(P, y) = d_{n+2} \geq \deg(Q, y) = d_{n+1} > 0$.

Theorem

The size of the evaluation cube of P and Q in y is

$$\frac{m d_{n+1}(1 + d_{n+1})}{2},$$

where m is the smallest power of 2 such that

$$m > (d_{n+2} + d_{n+1}) \left(d_1 + \sum_{i=2}^{n} d_i \prod_{j=1}^{i-1} (d_{n+2} d_j + d_{n+1} d_j + 1) \right).$$

Space Complexity

Corollary

When $d_{n+2} = d_{n+1} = \cdots = d_1 = d$, the FFT size is $\Theta(2^n d^{2n})$ and the size of the evaluation cube is $\Theta(2^n d^{2n+2})$.

n	d	FFT Degree	Cube Size	n	d	FFT Degree	Cube Size
1	80	14	203MB	3	6	19	42MB
1	100	15	632MB	3	8	22	576MB
1	120	15	908MB	3	10	23	1760MB
1	140	16	2468MB	4	5	23	480MB
2	15	18	120MB	4	6	25	2688MB
2	20	20	840MB	5	3	22	96MB
2	25	21	2600MB	5	4	26	2560MB

Algebraic Complexity

Theorem

The number of field operations in $\mathbb{Z}_p$ for computing the evaluation cube of P and Q in y is

$$O(m \log m(d_{n+1} + d_{n+2} + 2) + m(d_{n+1}^2 + d_{n+2}^2 + d_{n+1}d_{n+2}))$$

where m is the smallest power of 2 such that

$$m > (d_{n+2} + d_{n+1}) \left(d_1 + \sum_{i=2}^{n} d_i \prod_{j=1}^{i-1} (d_{n+2}d_j + d_{n+1}d_j + 1) \right).$$

When $d_{n+2} = d_{n+1} = \cdots = d_1 = d$, the cost to build the FFT based evaluation cube is $O(2^n d^{2n+2})$.

Subresultant Chain by Evaluation/Interpolation

Different Strategies

- FFT based technique
  - Fourier prime limitation
  - valid grid
  - translation
- subproduct tree technique

FFT scube on the GPU

- Coarse-grained construction
- Fine-grained construction

Brown's Subresultant Chain Algorithm

Input : Poly. $P, Q \in \mathbf{k}[y]$ s.t. $\deg(P) \geq \deg(Q) > 0$
Output : The subresultant chain of P and Q

1 $S_i \leftarrow 0$ for $0 \leq i < \deg(Q)$;
2 $B \leftarrow \mathtt{prem}(P, -Q, y)$, $A \leftarrow Q$, $\alpha \leftarrow \deg(P) - \deg(Q)$;
3 **while** $B \neq 0$ **do**
4 $\quad d \leftarrow \deg(A)$, $e \leftarrow \deg(B)$, $\delta \leftarrow d - e$;
5 $\quad S_{d-1} \leftarrow B$;
6 $\quad S_e \leftarrow \mathtt{lc}(A)^{\alpha(1-\delta)}\, \mathtt{lc}(B)^{\delta-1}\, B$;
7 $\quad$ **if** $e = 0$ **then break**;
8 $\quad B \leftarrow \mathtt{lc}(A)^{-\alpha\delta-1}\, \mathtt{prem}(A, -B, y)$, $A \leftarrow S_e$, $\alpha \leftarrow 1$;
9 **return** S_i *for* $0 \leq i < \deg(Q)$;

Approach (I) : Coarse-grained Implementation

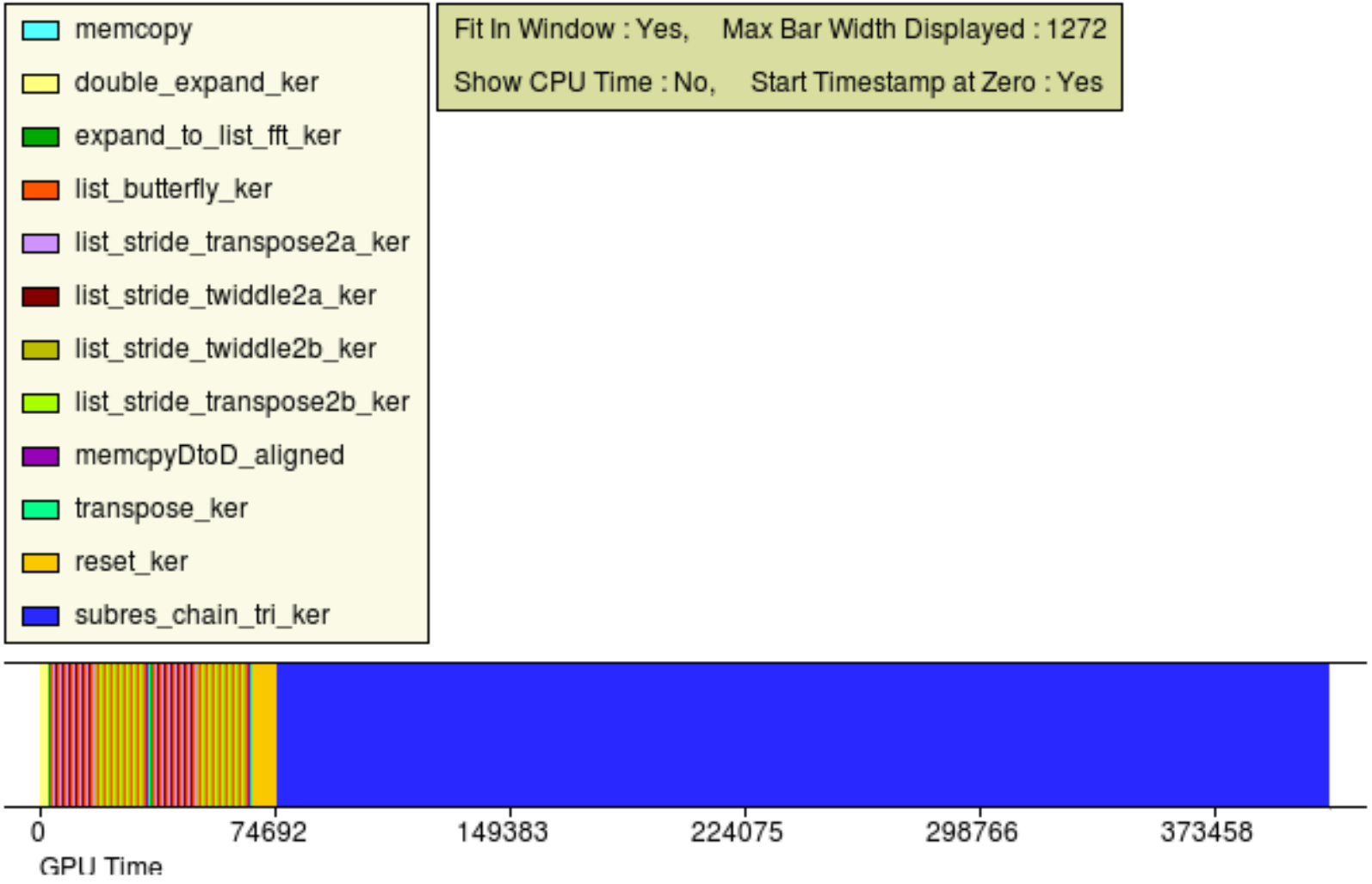
Main Idea

Each CUDA thread runs a univariate Brown's subresultant algorithm.

Remarks

- Simple and always works.
- The number of threads is bounded by the FFT size m, which is $\Theta(d^2)$ for a random dense square system of partial degree d.

Profiling Coarse-grained Implementation



Approach (II) : Fine-grained Implementation

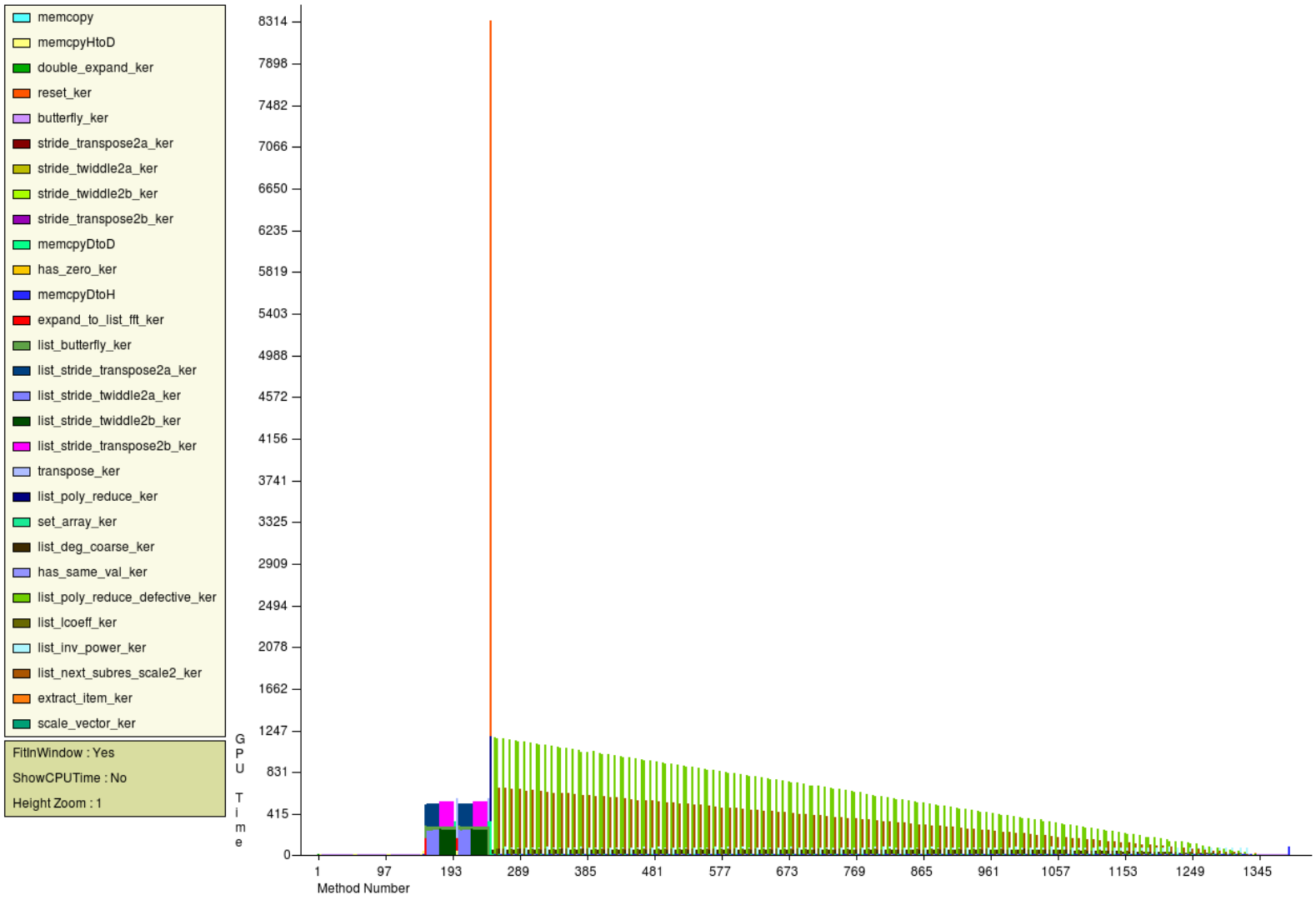
Main Idea

Compute subresultants level by level and parallelize the pseudo-division in the Brown's subresultant algorithm.

Remarks

- Complicate.
- The total number of threads is $\Theta(d^3)$.
- Require the following further assumption:
  The degree sequences of all images (P_j, Q_j) are the same.

Profiling Fine-grained Implementation



Computing Resultants

d	t_0	t_1	t_1/t_0
30	0.23	0.29	1.3
40	0.23	0.43	1.9
50	0.27	1.14	4.2
60	0.27	1.53	5.7
70	0.31	3.95	12.7
80	0.32	4.88	15.3
90	0.35	5.95	17.0
100	0.50	19.10	38.2
110	0.53	17.89	33.8
120	0.58	19.72	34.0

Bivariate dense polynomials of total degree d.

d	t_0	t_1	t_1/t_0
8	0.23	0.76	3.3
9	0.24	0.85	3.5
10	0.25	0.98	3.9
11	0.24	1.10	4.6
12	0.30	4.96	16.5
13	0.31	5.52	17.8
14	0.32	6.07	19.0
15	0.78	8.95	11.5
16	0.65	31.65	48.7
17	0.66	34.55	52.3
18	3.46	47.54	13.7
19	0.73	51.04	69.9
20	0.75	43.12	57.5

Trivariate dense polynomials of total degree d.

- t_0, GPU fft code
- t_1, CPU fft code
- Nvidia Tesla C2050

Preconditioning

In our implementation, linear translations are used to enlarge the feasibility of FFT based evaluations.

Theorem

Let m be the FFT size and let f be a polynomial of degree at most $\frac{p}{2m}$. Then the number of valid linear translations $\phi_a : x \mapsto x + a$ for f is at least $\frac{p}{2}$.

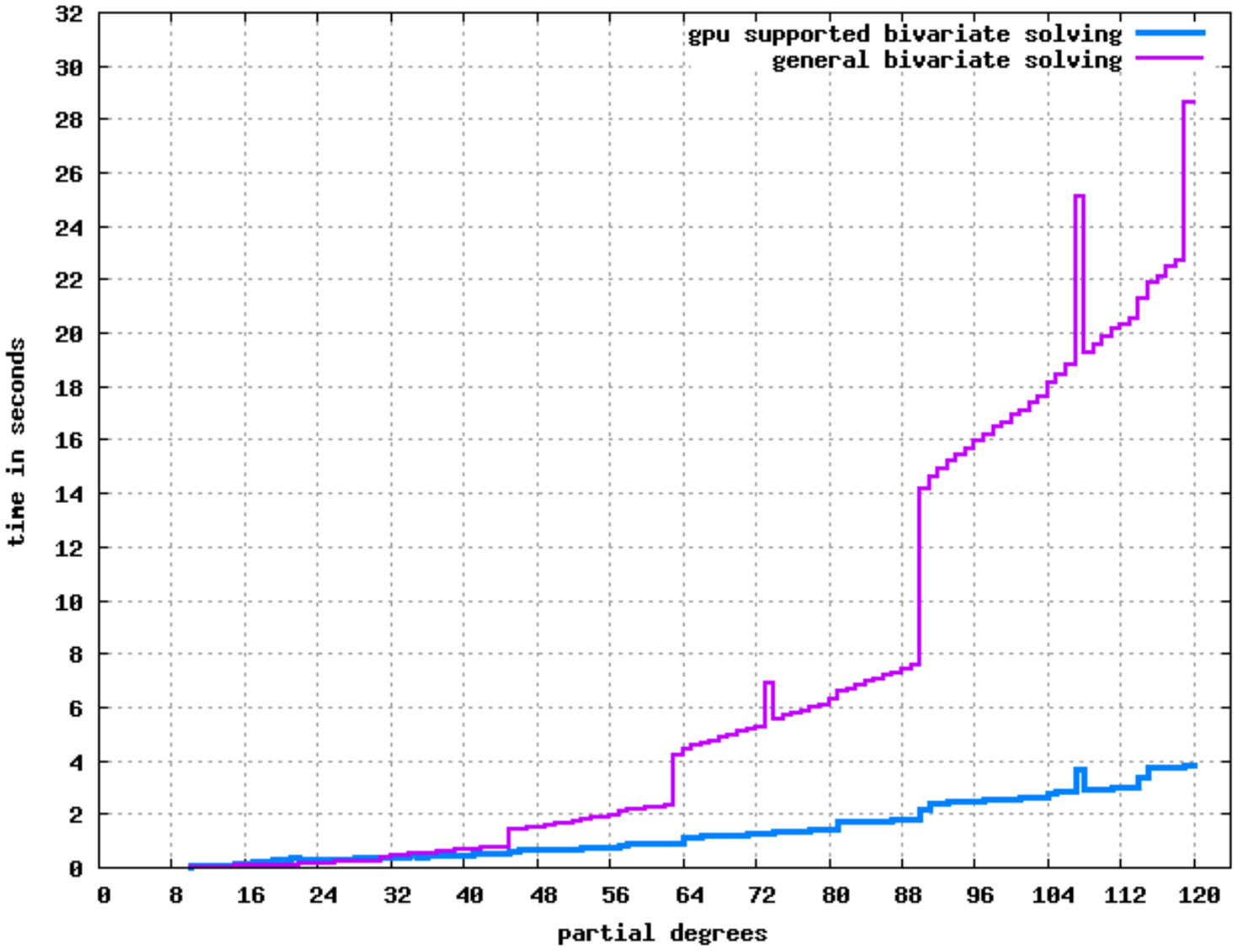
An Application to Solving Bivariate Systems

Generic Bivariate Solver

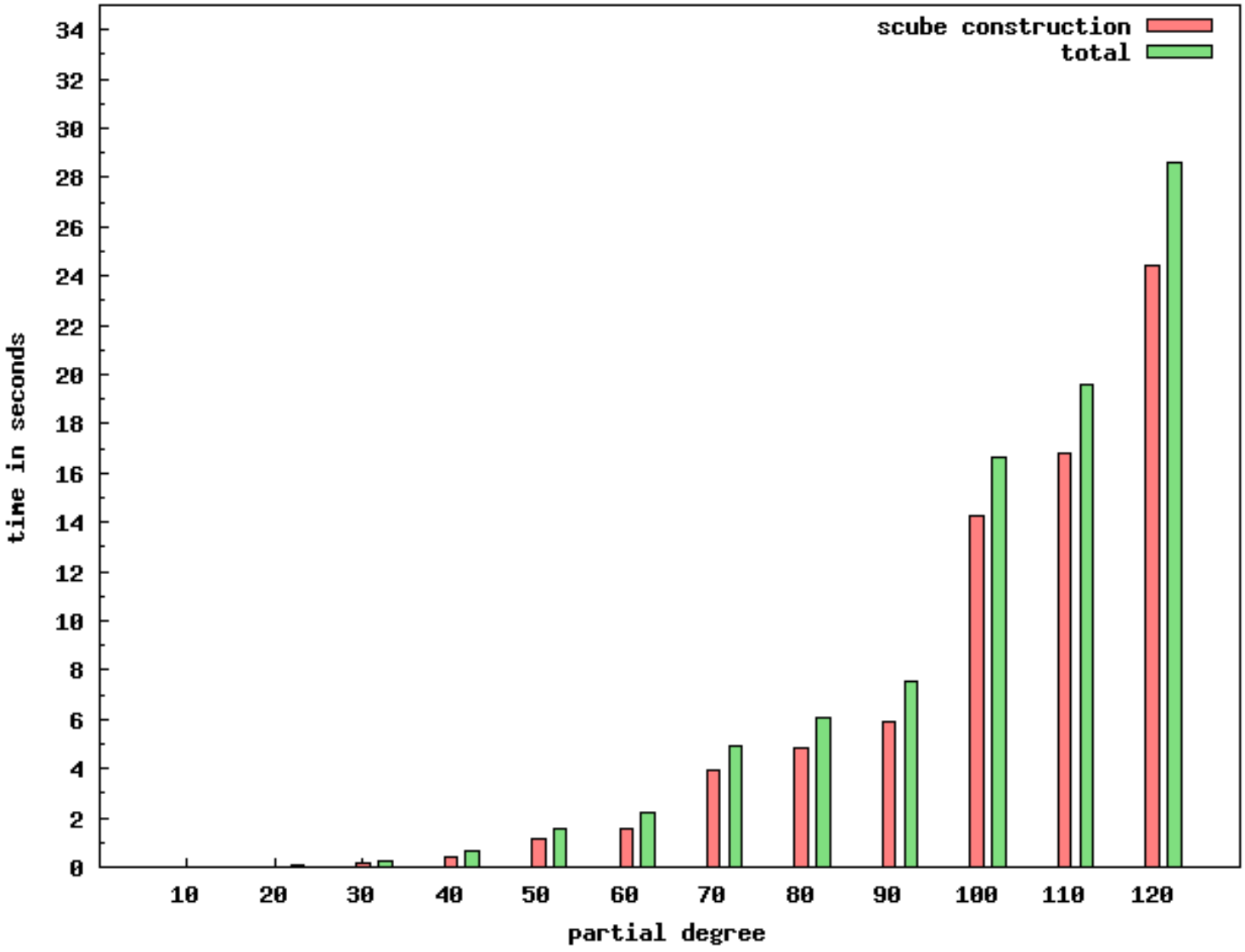
Algorithm 1: ModularGenericSolve2(F_1, F_2)

Compute the subresultant chain S of F_1, F_2 in y by value;
$R \leftarrow \texttt{sqrfree}(S_0)$, $result \leftarrow \emptyset$, $i \leftarrow 1$;
while $R \notin \mathbf{k}$ **and** $i \leq \deg(F_2, y)$ **do**
  while $i \leq \deg(F_2, y)$ **do**
    Let S_j be the regular subresultant with
    $i \leq j \leq \deg(F_2, y)$ being minimal;
    if $\mathrm{lc}(S_j, y) \equiv 0 \bmod R$ **then** $i \leftarrow i + 1$;
    else break;
  if $i > \deg(F_2, y)$ **then return** $result \cup \{(R, F_1)\}$;
  $G \leftarrow \gcd(R, \mathrm{lc}(S_j, y))$;
  if $G \in \mathbf{k}$ **then return** $result \cup \{(R, S_j)\}$;
  $result \leftarrow result \cup \{(R \text{ quo } G, S_j)\}$, $R \leftarrow G$, $i \leftarrow j + 1$;
return $result$;

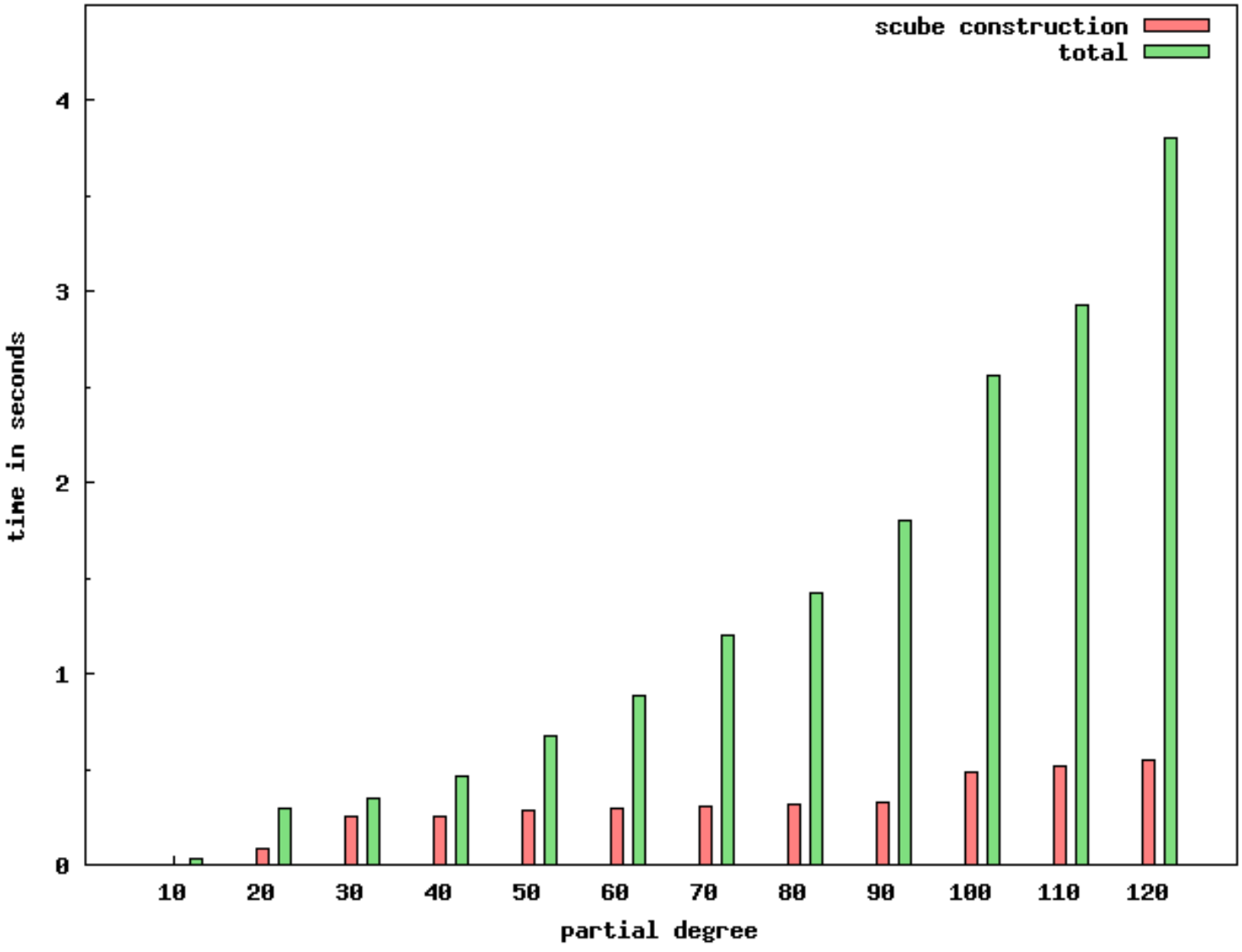
Bivariate Solver



Bivariate Solver on the CPU



Bivariate Solver on the GPU



Solving Bivariate Systems in Seconds

d	t_0(gpu)	t_1(total)	t_2 (cpu)	t_3 (total)	t_2/t_0	t_3/t_1
30	0.25	0.35	0.14	0.25	0.6	0.7
40	0.25	0.46	0.42	0.64	1.7	1.4
50	0.28	0.67	1.14	1.56	4.1	2.3
60	0.29	0.88	1.54	2.20	5.3	2.5
70	0.31	1.20	3.94	4.94	12.7	4.1
80	0.32	1.42	4.84	6.06	15.1	4.3
90	0.33	1.80	5.94	7.54	18.0	4.2
100	0.48	2.56	14.23	16.66	29.7	6.5
110	0.52	2.93	16.78	19.58	32.1	6.7
120	0.55	3.80	24.41	28.60	44.4	7.5

- d : total degree of the input polynomial
- t_0 : GPU FFT based scube construction
- t_1 : total time for solving with GPU code
- t_2 : CPU FFT base scube construction
- t_3 : total time for solving without CPU code

Summary

- For input of degree d, generic bivariate system solver has two major components
  - the subresultant chain construction: $O(d^4)$,
  - the univariate gcd computations: $O(d^{2+\epsilon})$.

  where d is the total degree of the input polynomials.
- The subresultant chain construction has been improved by a factor of (up to) 44 on the GPU.
- The current dominate part is the univariate gcd computation.

Acknowledgments

- I am deeply grateful to my supervisor, Marc Moreno Maza for his support, guidance, ideas and encouragement during the past four years. Without his help, this thesis would not exist.
- I want to thank my colleagues François Lemaire, Yuzhen Xie, Xin Li, Oleg Golubitsky, Changbo Chen, Sardar Anisul Haque, Liyun Li, Paul Vrbik, Rong Xiao for providing me help and for sharing their knowledge.
- I want to thank my colleagues at Maplesoft, in particular Jürgen Gerhard and Clare So, for our cooperation on the `RegularChains` and `modpn` libraries and for their help during my internship at Maplesoft.
- Many thanks to the members of my committee Mark Daley, Jan Minac, Jean-Louis Roch and Roberto Solis-Oba for their reading of this thesis and comments.
- Many thanks to all professors and students at ORCCA, where I spent this wonderful period of my life.

Specifications of GPUs

GPU	GTX 285	Tesla C2050
Compute Capability	1.3	2.0
Multiprocessors	30	14
Cores	240	448
Clock Rate	1.15G GHz	1.15 GHz
Memory Bandwidth	159 GB/sec	144 GB/sec
Single MADD	1062.7 GFLOPS	??
Double FMUL	1030.4 GFLOPS	515.2 GFLOPS
Double Floating Point	partially	fully
Global Memory	1GB	3GB
Shared Memory	16KB	48KB or 16KB
L1 Cache	none	48KB or 16KB
L2 Cache	none	768KB
Concurrent Kernels	no	up to 16

The GFLOPS of our CPU Intel Core Quad Q9400 is 7.48.

Triangular Decomposition in Positive Dimension

- Given a polynomial system F consisting of two polynomials with variable ordering $x > y > a > b > c > d > e > f$

$$F : \left| \begin{array}{rcl} a\,x + c\,y - e & = & 0 \\ b\,x + d\,y - f & = & 0 \end{array} \right.$$

- A triangular decomposition algorithm transforms F into a set $\{T\}$ of regular chains

$$T : \left| \begin{array}{r} (a\,d - b\,c)\,y + b\,e - a\,f \\ b\,x + d\,y - f \end{array} \right.$$

- The solution set is preserved during the transformation, i.e.

$$V(\mathtt{sat}(T)) = V(F).$$

Triangular Decomposition in Positive Dimension

- Given a polynomial system F consisting of two polynomials with variable ordering $x > y > a > b > c > d > e > f$

$$F : \left| \begin{array}{rcl} a\,x + c\,y - e & = & 0 \\ b\,x + d\,y - f & = & 0 \end{array} \right.$$

- A triangular decomposition algorithm transforms F into a set $\{T\}$ of regular chains

$$T : \left| \begin{array}{r} (a\,d - b\,c)\,y + b\,e - a\,f \\ b\,x + d\,y - f \end{array} \right.$$

- The solution set is preserved during the transformation, i.e.

$$V(\mathtt{sat}(T)) = V(F).$$

Two FFT formulas

Cooley-Tukey FFT

$$\mathrm{DFT}_{2^k} = \left(\prod_{i=1}^{k} \left(I_{2^{i-1}} \otimes \mathrm{DFT}_2 \otimes I_{2^{k-i}} \right) T_{n,i} \right) R_n$$

with the twiddle factor matrix $T_{n,i} = I_{2^{i-1}} \otimes D_{2,2^{k-i}}$ and the bit-reversal permutation matrix

$$R_n = (I_{n/2} \otimes L_2^2)(I_{n/2^2} \otimes L_2^4) \cdots (I_1 \otimes L_2^n).$$

Stockham FFT

$$\mathrm{DFT}_{2^k} = \prod_{i=0}^{k-1} \underbrace{(\mathrm{DFT}_2 \otimes I_{2^{k-1}})}_{\text{butterfly}} \underbrace{(D_{2,2^{k-i-1}} \otimes I_{2^i})}_{\text{twiddling}} \underbrace{(L_2^{2^{k-i}} \otimes I_{2^i})}_{\text{reordering}}$$

Break pseudo-divisions

Example

Let $f = a_3x^3 + a_2x^2 + a_1x + a_0$ and $g = b_2x^2 + b_1x + b_0$. To obtain the pseudo-remainder $\mathtt{prem}(f, -g, x)$ of f and g, we compute

1. $h_2 = -b_2 f + a_3 x g = c_2x^2 + c_1x + c_0$,
2. $h_1 = -b_2 h_2 + c_2 g = d_1x + b_0$.

Alternatively, we compute

$$\text{(S1)} \quad c_2 = \begin{vmatrix} a_3 & a_2 \\ b_2 & b_1 \end{vmatrix} \quad c_1 = \begin{vmatrix} a_3 & a_1 \\ b_2 & 0 \end{vmatrix} \quad c_0 = \begin{vmatrix} a_3 & a_0 \\ b_2 & 0 \end{vmatrix}$$

$$\text{(S2)} \quad d_1 = \begin{vmatrix} c_2 & c_1 \\ b_2 & b_1 \end{vmatrix} \quad d_0 = \begin{vmatrix} c_2 & c_0 \\ b_2 & b_0 \end{vmatrix}$$

One can do a list of pseudo-divisions with the same input degrees.

Univariate polynomial multiplication over finite fields

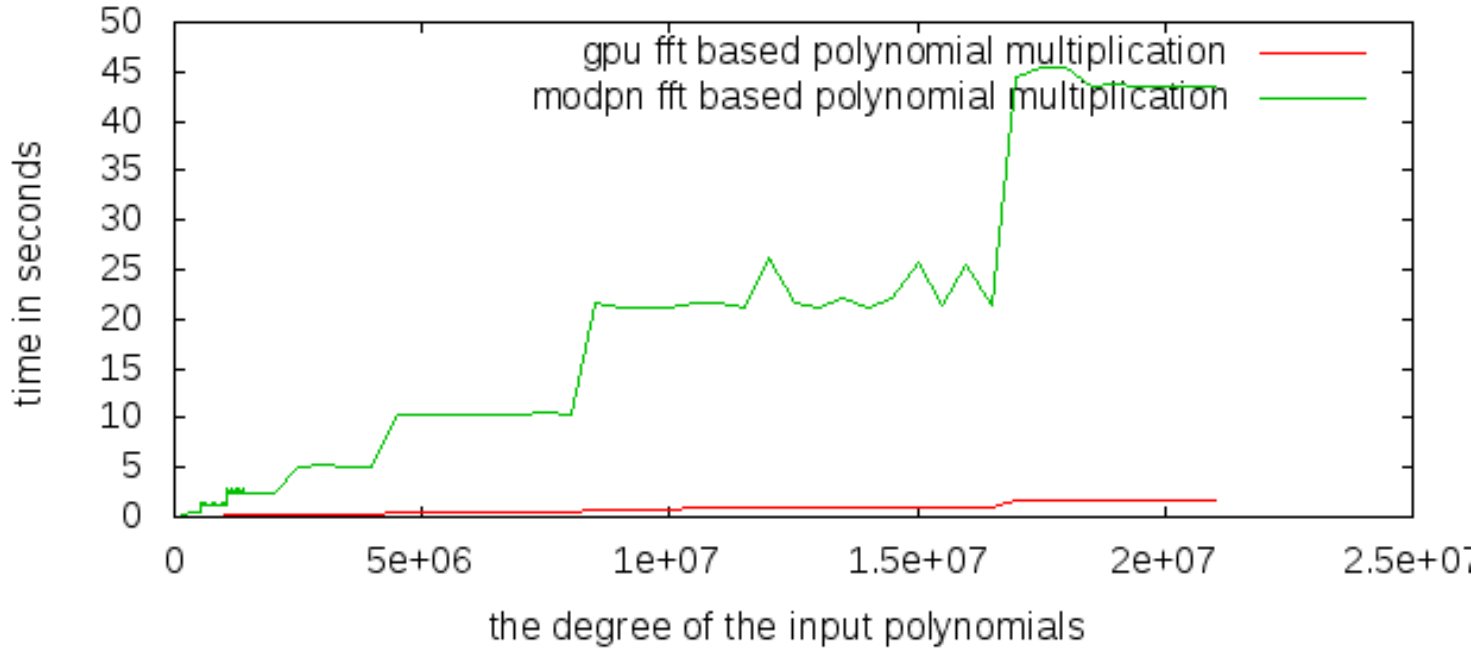


Figure: FFT-based polynomial multiplication on GPU and CPU

Generalized CUDA kernels

To realize bivariate FFTs, we implemented some more CUDA kernels:

$I_m \otimes \mathrm{DFT}_{2^k}$

- $(I_m \otimes \mathrm{DFT}_2 \otimes I_{2^{k-1}})$, list_butterfly_kernel
- $(I_m \otimes D_{2,2^{k-i-1}} \otimes I_{2^i})$, list_twiddling_kernel
- $(I_m \otimes L_2^{2^{k-i}} \otimes I_{2^i})$, list_reordering_kernel

$\mathrm{DFT}_{2^k} \otimes I_m$

- $(\mathrm{DFT}_2 \otimes I_{2^{k-1}} \otimes I_m)$, ext_butterfly_kernel
- $(D_{2,2^{k-i-1}} \otimes I_{2^i} \otimes I_m)$, ext_twiddling_kernel
- $(L_2^{2^{k-i}} \otimes I_{2^i} \otimes I_m)$, ext_reordering_kernel

List of bivariate FFTs $I_q \otimes \mathrm{DFT}_{m \times n}$ can be derived as well.

Bivariate multiplication

nx(ny)	Kronecker + modpn fft	modpn 2d fft	fftmul2
100	0.020	0.020	0.040
200	0.080	0.080	0.040
300	0.440	0.420	0.070
400	0.500	0.530	0.070
600	2.690	1.980	0.150
800	2.430	1.970	0.140
1000	2.950	2.430	0.160
1200	11.950	10.220	0.460
1400	10.810	8.150	0.470
1600	10.330	8.120	0.490
1800	10.260	8.130	0.500
2000	11.970	9.930	0.520
2200	45.300	35.470	1.780
2400	53.570	44.430	1.820

Figure: Bivariate polynomial multiplication with random dense input polynomials, timing in seconds. The data transfer between device and host is counted. The threshold is 100$\sim$200 for this machine and GPU.

$V(T)$ and $V(\mathtt{sat}(T))$

- Given ideals $I, J \subseteq \mathbb{Q}[x_1, \ldots, x_n]$, we have

$$V(I) = \{x \in \mathbb{C}^n \mid f(x) = 0 \text{ for all } f \in I\}$$
$$V(I \cap J) = V(I) \cup V(J).$$

- Since $\mathtt{sat}(T) = \langle u\,y + v, x\,y - 1\rangle$,

$$V(\mathtt{sat}(T)) = \{(x, y, u, v) \in \mathbb{C}^4 \mid x\,y = 1, u\,y + v = 0\}.$$

- Since $\langle T\rangle = \langle u\,y + v, x\,y - 1\rangle \cap \langle u, v\rangle$,

$$V(T) = V(\mathtt{sat}(T)) \cup \{(x, y, u, v) \in \mathbb{C}^4 \mid u = 0, v = 0\}.$$

- $\mathcal{O} = (0, 0, 0, 0) \in V(T)$, but $\mathcal{O} \notin V(\mathtt{sat}(T))$.

$V(T)$ and $V(\mathtt{sat}(T))$

- Given ideals $I, J \subseteq \mathbb{Q}[x_1, \ldots, x_n]$, we have

$$V(I) = \{x \in \mathbb{C}^n \mid f(x) = 0 \text{ for all } f \in I\}$$
$$V(I \cap J) = V(I) \cup V(J).$$

- Since $\mathtt{sat}(T) = \langle u\,y + v, x\,y - 1\rangle$,

$$V(\mathtt{sat}(T)) = \{(x, y, u, v) \in \mathbb{C}^4 \mid x\,y = 1, u\,y + v = 0\}.$$

- Since $\langle T\rangle = \langle u\,y + v, x\,y - 1\rangle \cap \langle u, v\rangle$,

$$V(T) = V(\mathtt{sat}(T)) \cup \{(x, y, u, v) \in \mathbb{C}^4 \mid u = 0, v = 0\}.$$

- $\mathcal{O} = (0, 0, 0, 0) \in V(T)$, but $\mathcal{O} \notin V(\mathtt{sat}(T))$.

$V(T)$ and $V(\mathtt{sat}(T))$

- Given ideals $I, J \subseteq \mathbb{Q}[x_1, \ldots, x_n]$, we have

$$V(I) = \{x \in \mathbb{C}^n \mid f(x) = 0 \text{ for all } f \in I\}$$
$$V(I \cap J) = V(I) \cup V(J).$$

- Since $\mathtt{sat}(T) = \langle u\,y + v, x\,y - 1\rangle$,

$$V(\mathtt{sat}(T)) = \{(x, y, u, v) \in \mathbb{C}^4 \mid x\,y = 1, u\,y + v = 0\}.$$

- Since $\langle T\rangle = \langle u\,y + v, x\,y - 1\rangle \cap \langle u, v\rangle$,

$$V(T) = V(\mathtt{sat}(T)) \cup \{(x, y, u, v) \in \mathbb{C}^4 \mid u = 0, v = 0\}.$$

- $\mathcal{O} = (0, 0, 0, 0) \in V(T)$, but $\mathcal{O} \notin V(\mathtt{sat}(T))$.

$V(T)$ and $V(\mathtt{sat}(T))$

- Given ideals $I, J \subseteq \mathbb{Q}[x_1, \ldots, x_n]$, we have

$$V(I) = \{x \in \mathbb{C}^n \mid f(x) = 0 \text{ for all } f \in I\}$$
$$V(I \cap J) = V(I) \cup V(J).$$

- Since $\mathtt{sat}(T) = \langle u\,y + v, x\,y - 1\rangle$,

$$V(\mathtt{sat}(T)) = \{(x, y, u, v) \in \mathbb{C}^4 \mid x\,y = 1, u\,y + v = 0\}.$$

- Since $\langle T\rangle = \langle u\,y + v, x\,y - 1\rangle \cap \langle u, v\rangle$,

$$V(T) = V(\mathtt{sat}(T)) \cup \{(x, y, u, v) \in \mathbb{C}^4 \mid u = 0, v = 0\}.$$

- $\mathcal{O} = (0, 0, 0, 0) \in V(T)$, but $\mathcal{O} \notin V(\mathtt{sat}(T))$.

A Remark

- A straightforward generalization of primitivity is not enough.
- Consider $T = \{t_1 = uy + v, t_2 = vx + u\}$. Then
  - t_1 is primitive over $\Bbbk[u, v]$;
  - t_2 is primitive over $\Bbbk[u, v, y]$.
- However, $\mathtt{sat}(T)$ is strictly larger than $\langle T \rangle$.

The Question

- **Proposition**: $f = a_d x^d + \cdots + a_0 \in \mathbb{A}[x]$ is primitive iff

$$\langle f \rangle : a_d^\infty = \langle f \rangle,$$

where $\mathbb{A}$ is a UFD.

- Restate this proposition as: For each $f \in \mathbf{k}[x_1, \ldots, x_n]$

$$\mathtt{sat}(f) = \langle f \rangle \iff f \text{ is primitive in its main variable.}$$

- When does $\langle T \rangle$ equal $\mathtt{sat}(T)$? **Primitive regular chains**?

Example

Consider $F_1 = x^2 + y + 1$ and $F_2 = x + y^2 + 1$ in $\mathbf{k}[x, y]$. The common solution of F_1 and F_2 can be encoded by a triangular set

$$\begin{aligned} A(x) &= x^4 + 2\,x^2 + x + 2 \\ B(x, y) &= y + x^2 + 1. \end{aligned}$$

- $A(x)$ is the resultant of F_1 and F_2 in y,
- $B(x, y)$ is the gcd of F_1 and F_2 modulo the relation $A(x) = 0$.

Structural formulas for DFT matrices

Factorization of DFT matrices

Most FFTs (Cooley-Tukey, Stockham, etc.) can be derived from

$$\mathrm{DFT}_{pq} = (\mathrm{DFT}_p \otimes I_q) D_{p,q} (I_p \otimes \mathrm{DFT}_q) L^{pq}_p \tag{2}$$

where $D_{p,q}$ is a diagonal matrix of twiddle factors and L^{pq}_p is a stride permutation matrix.

Example

$$\mathrm{DFT}_8 = \begin{cases} (\mathrm{DFT}_2 \otimes I_4) D_{2,4} (I_2 \otimes \mathrm{DFT}_4) L^8_2 \\ (\mathrm{DFT}_4 \otimes I_2) D_{4,2} (I_4 \otimes \mathrm{DFT}_2) L^8_4 \end{cases}$$

Bivariate Case

1. Compute the degree bound $m = 2^\ell$ with

$$\deg_x \texttt{resultant}(P, Q, y) < m.$$

2. Evaluate P, Q at $x = \omega^j$ for $j = 0 \cdots m-1$,

$$P_j = I_{p+1} \otimes \mathrm{DFT}_m^\omega(P) \text{ and } Q_j = I_{q+1} \otimes \mathrm{DFT}_m^\omega(Q),$$

   with a well-chosen m-primitive root of unity ω.

3. For each evaluation (P_j, Q_j), compute

$$R_j = \texttt{subres}(P_j, Q_j, y).$$

4. The set $\mathcal{R} = \{R_0, \ldots, R_{m-1}\}$ is called an scube of P, Q in y.

Polynomial Subresultant Chain

Setting

- $P, Q \in \mathbb{Z}_p[x_1, \ldots, x_n, y]$ with $\deg(P, y) \geq q = \deg(Q, y) > 0$
- the subresultant chain of P, Q is a polynomial sequence

$$\texttt{subres}(P, Q, y) = (S_{q-1}, \ldots, S_1, S_0),$$

computed by a subresultant chain algorithm,

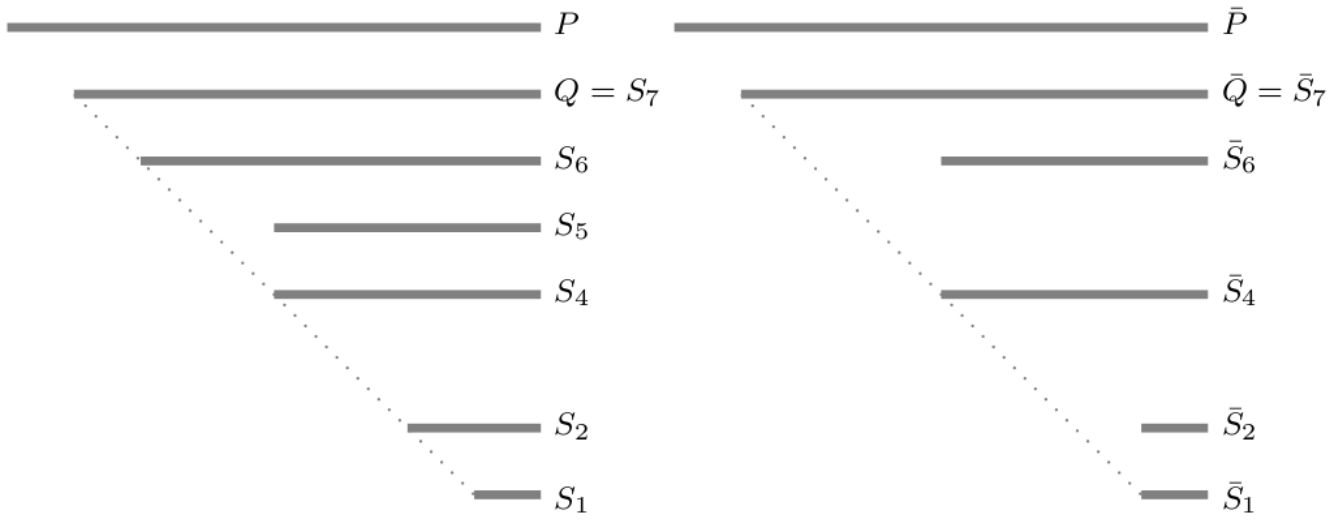
- The specialization property,

$$\pi(\texttt{subres}_j(P, Q, y)) = \texttt{subres}_j(\pi(P), \pi(Q), y),$$

if $\deg(\pi(P), y) = \deg(P, y)$, and $\deg(\pi(Q), y) = \deg(Q, y)$ where $\pi : \mathbb{Z}_p[x_1, \ldots, x_n] \to \mathbb{Z}_p[x_1, \ldots, x_n]$ is a ring homomorphism,

Subresultants and Regular GCD

On the left, P and Q have five nonzero subresultants.



On the right, $\bar{P}$ and $\bar{Q}$ have four nonzero subresultants modulo $\mathtt{sat}(T)$. Our algorithm says that S_1 is a regular GCD of P, Q modulo $\mathtt{sat}(T)$ if $\mathtt{init}(S_1), \mathtt{init}(S_4)$ are regular modulo $\mathtt{sat}(T)$.