Optimizing and and Parallelizing the Modular GCD Algorithm

Matthew Gibson   Michael Monagan

Centre for Experimental and Constructive Mathematics
Simon Fraser University
British Columbia

PASCO 2015, Bath, England
July 10, 2015

Problem

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Problem

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Compute G modulo primes $p_1, p_2, \ldots$ and recover G using Chinese remaindering.

Problem

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Compute G modulo primes $p_1, p_2, \ldots$ and recover G using Chinese remaindering.

Let $\bar{A} = A/G$ and $\bar{B} = B/G$ be the cofactors.
Let $A = \sum_{i=0}^{da} a_i(x_2, ..., x_n) x_1^i$.
Let $B = \sum_{i=0}^{db} b_i(x_2, ..., x_n) x_1^i$.
Let $G = \sum_{i=0}^{dg} g_i(x_2, ..., x_n) x_1^i$.
Let $t = \max_{i=0}^{dg} \#terms\ g_i$.

Interpolate $g_i(x_2, ..., x_n)$ modulo p from $2t + \delta$ univariate images in $\mathbb{Z}_p[x_1]$ using smooth prime p.

Problem

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Compute G mod $p_1, p_2, \ldots$ and recover G using Chinese remaindering.

Let $\bar{A} = A/G$ and $\bar{B} = B/G$ be the cofactors.
Let $A = \sum_{i=0}^{da} a_i(x_2, ..., x_n) x_1^i$. $\quad CA = GCD(a_i(x_2, ..., x_n))$.
Let $B = \sum_{i=0}^{db} b_i(x_2, ..., x_n) x_1^i$. $\quad CB = GCD(b_i(x_2, ..., x_n))$.
Let $G = \sum_{i=0}^{dg} g_i(x_2, ..., x_n) x_1^i$. $\quad CG = GCD(CA, CB)$.
Let $t = \max_{i=0}^{dg} \#terms\ g_i$. $\quad \Gamma = GCD(a_{da}, b_{db})$.

Observation: Most of the time is recursive GCDs in $n-1$ variables and evaluation and interpolation not GCD in $\mathbb{Z}_p[x_1]$.

Bivariate Images

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Let $A = \sum_i a_{i,j}(x_3, ..., x_n)x_1^i x_2^j$. $CA = GCD(a_i(x_3, ..., x_n))$.
Let $B = \sum_i b_{i,j}(x_3, ..., x_n)x_1^i x_2^j$. $CB = GCD(b_i(x_3, ..., x_n))$.
Let $G = \sum_i g_{i,j}(x_3, ..., x_n)x_1^i x_2^j$. $CG = GCD(CA, CB)$.
Let $s = \max_{i,j} \#terms\ g_{i,j}$. $\Gamma = GCD(LC(A), LC(B))$.

Interpolate $g_i(x_3, ..., x_n)$ modulo p from $2s + \delta$ bivariate images in $\mathbb{Z}_p[x_1, x_2]$ using smooth prime p – increased cost but

Bivariate Images

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$.

Let $A = \sum_i a_{i,j}(x_3, \ldots, x_n) x_1^i x_2^j$. $CA = GCD(a_i(x_3, \ldots, x_n))$.
Let $B = \sum_i b_{i,j}(x_3, \ldots, x_n) x_1^i x_2^j$. $CB = GCD(b_i(x_3, \ldots, x_n))$.
Let $G = \sum_i g_{i,j}(x_3, \ldots, x_n) x_1^i x_2^j$. $CG = GCD(CA, CB)$.
Let $s = \max_{i,j} \# terms\ g_{i,j}$. $\Gamma = GCD(LC(A), LC(B))$.

Interpolate $g_i(x_3, \ldots, x_n)$ modulo p from $2s + \delta$ bivariate images in $\mathbb{Z}_p[x_1, x_2]$ using smooth prime p – increased cost but

- Usually $s \ll t$ which reduces evaluation and interpolation cost.

Bivariate Images

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Let $A = \sum_i a_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $\quad CA = GCD(a_i(x_3, ..., x_n))$.
Let $B = \sum_i b_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $\quad CB = GCD(b_i(x_3, ..., x_n))$.
Let $G = \sum_i g_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $\quad CG = GCD(CA, CB)$.
Let $s = \max_{i,j} \# terms\ g_{i,j}$. $\quad \Gamma = GCD(LC(A), LC(B))$.

Interpolate $g_i(x_3, ..., x_n)$ modulo p from $2s + \delta$ bivariate images in $\mathbb{Z}_p[x_1, x_2]$ using smooth prime p – increased cost but

- Usually $s \ll t$ which reduces evaluation and interpolation cost.
- Usually CA, CB, Γ are smaller so easier to compute.

Bivariate Images

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Let $A = \sum_i a_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $CA = GCD(a_i(x_3, ..., x_n))$.
Let $B = \sum_i b_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $CB = GCD(b_i(x_3, ..., x_n))$.
Let $G = \sum_i g_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $CG = GCD(CA, CB)$.
Let $s = \max_{i,j} \#terms\ g_{i,j}$. $\Gamma = GCD(LC(A), LC(B))$.

Interpolate $g_i(x_3, ..., x_n)$ modulo p from $2s + \delta$ bivariate images in $\mathbb{Z}_p[x_1, x_2]$ using smooth prime p – increased cost but

- Usually $s \ll t$ which reduces evaluation and interpolation cost.
- Usually CA, CB, Γ are smaller so easier to compute.
- Increases parallelism in interpolation.

Bivariate Images

Compute $G = \mathrm{GCD}(A, B)$ in $\mathbb{Z}[x_1, x_2, ..., x_n]$.

Let $A = \sum_i a_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $CA = GCD(a_i(x_3, ..., x_n))$.
Let $B = \sum_i b_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $CB = GCD(b_i(x_3, ..., x_n))$.
Let $G = \sum_i g_{i,j}(x_3, ..., x_n) x_1^i x_2^j$. $CG = GCD(CA, CB)$.
Let $s = \max_{i,j} \#terms\ g_{i,j}$. $\Gamma = GCD(LC(A), LC(B))$.

Interpolate $g_i(x_3, ..., x_n)$ modulo p from $2s + \delta$ bivariate images in $\mathbb{Z}_p[x_1, x_2]$ using smooth prime p – increased cost but

- Usually $s \ll t$ which reduces evaluation and interpolation cost.
- Usually CA, CB, Γ are smaller so easier to compute.
- Increases parallelism in interpolation.

1. Optimize serial bivariate Gcd computation.
2. For $n > 2$ parallelized (Cilk C) evaluation and interpolation.
3. Benchmark against Maple and Magma.

Bivariate Gcd computation.

Input $A, B \in \mathbb{Z}_p[y][x]$. Output $G = GCD(A, B)$, $\bar{A}$ and $\bar{B}$.

Trial division method. (Maple, Magma)
Interpolate y in G from univariate images in $\mathbb{Z}_p[x]$ **incrementally** until $G(x, y)$ does not change.
Test if $G|A$ and $G|B$. If yes output $G, \bar{A} = A/G, \bar{B} = B/G$.

Bivariate Gcd computation.

Input $A, B \in \mathbb{Z}_p[y][x]$. Output $G = GCD(A, B)$, $\bar{A}$ and $\bar{B}$.

Trial division method. (Maple, Magma)
Interpolate y in G from univariate images in $\mathbb{Z}_p[x]$ **incrementally** until $G(x, y)$ does not change.
Test if $G|A$ and $G|B$. If yes output $G, \bar{A} = A/G, \bar{B} = B/G$.

Cofactor recovery method. (Brown 1971)

Interpolate y in $G, \bar{A}, \bar{B}$ from univariate images
$g_i = G(\alpha_i, x), \bar{a}_i = A(\alpha_i, x)/g_i, \bar{b}_i = B(\alpha_i, x)/g_i$ in $\mathbb{Z}_p[x]$.
After k images we have

$$A - G\bar{A} \equiv 0 \pmod{M} \text{ and } B - G\bar{B} \equiv 0 \pmod{M}$$

where $M = (y - \alpha_1)(y - \alpha_2) \cdots (y - \alpha_k)$.
Stop when $k > \max(\deg_y A, \deg_y B, \deg_y G\bar{A}, \deg_y G\bar{B})$.

Cofactor recovery method for $\mathbb{Z}_p[y][x]$

Interpolate y in $G, \bar{A}, \bar{B}$ from univariate images
$g_i = G(\alpha_i, x), \bar{a}_i = A(\alpha_i, x)/g_i, \bar{b}_i = B(\alpha_i, x)/g_i$ in $\mathbb{Z}_p[x]$
in batches until one of $G, \bar{A}, \bar{B}$ **stabilizes**.

Case G stabilizes: obtain remaining images using univariate $\div$
$g_i = G(\alpha_i, x), \bar{a}_i = A(\alpha_i, x)/g_i, \bar{b}_i = B(\alpha_i, x)/g_i$
thus replacing the Euclidean algorithm with an evaluation.

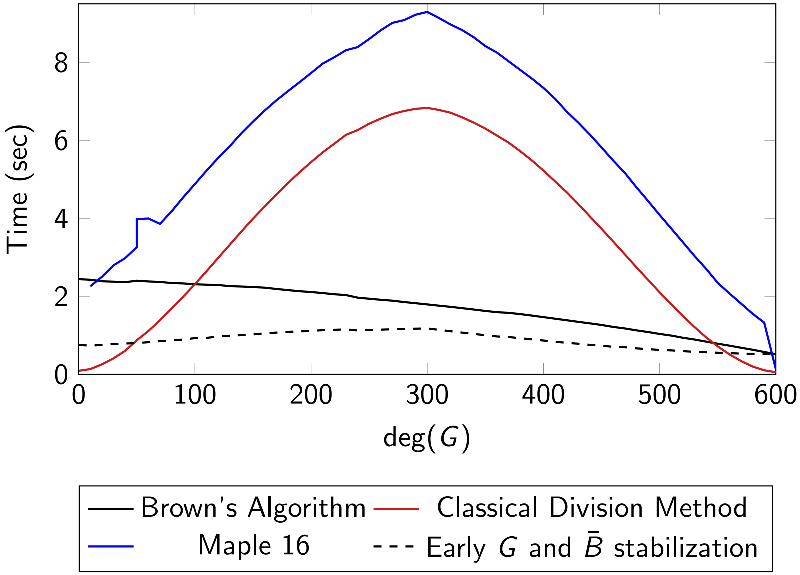
Cofactor recovery method for $\mathbb{Z}_p[y][x]$

Interpolate y in $G, \bar{A}, \bar{B}$ from univariate images
$g_i = G(\alpha_i, x), \bar{a}_i = A(\alpha_i, x)/g_i, \bar{b}_i = B(\alpha_i, x)/g_i$ in $\mathbb{Z}_p[x]$
in batches until one of $G, \bar{A}, \bar{B}$ **stabilizes**.

Case G stabilizes: obtain remaining images using univariate $\div$
$g_i = G(\alpha_i, x), \bar{a}_i = A(\alpha_i, x)/g_i, \bar{b}_i = B(\alpha_i, x)/g_i$
thus replacing the Euclidean algorithm with an evaluation.

Case $\bar{A}$ stabilizes: obtain remaining images using univariate $\div$
$\bar{a}_i = \bar{A}(\alpha_i, x), g_i = A(\alpha_i, x)/\bar{a}_i, \bar{b}_i = B(\alpha_i, x)/g_i$
thus replacing the Euclidean algorithm with an evaluation.

Figure : Image Division Optimizations



Using FFT with small roots of unity

For dense A, B in $\mathbb{Z}_p[x_n][x_1 \dots x_{n-1}]$ we evaluate and interpolate A and B in blocks of size j using a FFT of size j ($j = 2, 4, 8, 16, \dots$).
The idea:

Using FFT with small roots of unity

For dense A, B in $\mathbb{Z}_p[x_n][x_1 \dots x_{n-1}]$ we evaluate and interpolate A and B in blocks of size j using a FFT of size j ($j = 2, 4, 8, 16, \dots$).

The idea:

- $f \in \mathbb{Z}_p[x_n]$

Using FFT with small roots of unity

For dense A, B in $\mathbb{Z}_p[x_n][x_1 \ldots x_{n-1}]$ we evaluate and interpolate A and B in blocks of size j using a FFT of size j ($j = 2, 4, 8, 16, \ldots$).

The idea:

- $f \in \mathbb{Z}_p[x_n]$
- $j = 2^k$, small, such that $j \mid p - 1$

Using FFT with small roots of unity

For dense A, B in $\mathbb{Z}_p[x_n][x_1 \ldots x_{n-1}]$ we evaluate and interpolate A and B in blocks of size j using a FFT of size j ($j = 2, 4, 8, 16, \ldots$). The idea:

- $f \in \mathbb{Z}_p[x_n]$
- $j = 2^k$, small, such that $j \mid p - 1$
- $f^* \equiv f \bmod (x^j - \alpha_0^j)$

Using FFT with small roots of unity

For dense A, B in $\mathbb{Z}_p[x_n][x_1 \dots x_{n-1}]$ we evaluate and interpolate A and B in blocks of size j using a FFT of size j ($j = 2, 4, 8, 16, \dots$).

The idea:

- $f \in \mathbb{Z}_p[x_n]$
- $j = 2^k$, small, such that $j \mid p - 1$
- $f^* \equiv f \mod (x^j - \alpha_0^j)$
- Evaluate f^* using the FFT

Parallel experiments in Cilk C

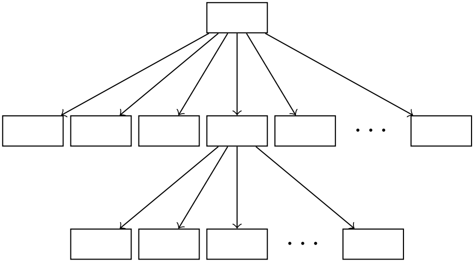
Using FFT with small roots of unity

For dense A, B in $\mathbb{Z}_p[x_n][x_1 \dots x_{n-1}]$ we evaluate and interpolate A and B in blocks of size j using a FFT of size j ($j = 2, 4, 8, 16, \dots$). The idea:

- $f \in \mathbb{Z}_p[x_n]$
- $j = 2^k$, small, such that $j \mid p - 1$
- $f^* \equiv f \bmod (x^j - \alpha_0^j)$
- Evaluate f^* using the FFT

Cilk is a `C/C++` extension for parallelism in computation. **Cilk** uses a fixed number of worker threads and a work-stealing algorithm, and two basic keywords: `cilk_spawn` and `cilk_sync`.
We implement with Cilk Plus by Intel.

Dense Polynomial Structure Recursive dense representation using arrays. Multivariate polynomials form a tree.
A, B in $\mathbb{Z}_p[x_1, x_2, x_3]$, monic, dense in total degree $d = 200$



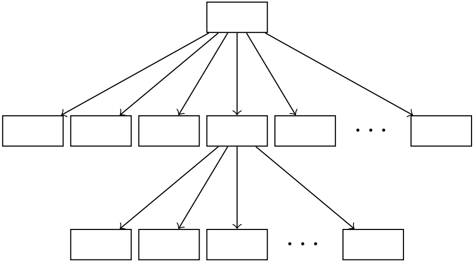
$A, B \in \mathbb{Z}_p[x_1, x_2, x_3]$

$\mathbb{Z}_p[x_2, x_3]$

$\mathbb{Z}_p[x_3]$

Parallel experiments in Cilk C

Dense Polynomial Structure Recursive dense representation using arrays. Multivariate polynomials form a tree.
A, B in $\mathbb{Z}_p[x_1, x_2, x_3]$, monic, dense in total degree $d = 200$



$A, B \in \mathbb{Z}_p[x_1, x_2, x_3]$

$\mathbb{Z}_p[x_2, x_3]$ $\quad d + 1 = 201$

$\mathbb{Z}_p[x_3]$ $\quad \frac{d^2+3d+2}{2} = 20503$

The number of terms in each input polynomial is 1.37 million, filling 10.5 MB of memory.

Parallel Implementation

Example: Call $\textsc{mgcd}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

Parallel Implementation

Example: Call $\text{MGCD}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

1. Allocate space for interpolants $G^*, \bar{A}^*, \bar{B}^*$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

Parallel Implementation

Example: Call $\textsc{MGCD}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

1. Allocate space for interpolants $G^*, \bar{A}^*, \bar{B}^*$ in $\mathbb{Z}_p[x_1, x_2, x_3]$
2. For $\lceil bnd/j \rceil$ batches: in parallel

Parallel Implementation

Example: Call $\textsc{MGCD}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

1. Allocate space for interpolants $G^*, \bar{A}^*, \bar{B}^*$ in $\mathbb{Z}_p[x_1, x_2, x_3]$
2. For $\lceil bnd/j \rceil$ batches: in parallel
    1. Evaluate j images of the inputs into new space in $\mathbb{Z}_p[x_1, x_2]$

Parallel Implementation

Example: Call $\textsc{MGCD}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

1. Allocate space for interpolants $G^*, \bar{A}^*, \bar{B}^*$ in $\mathbb{Z}_p[x_1, x_2, x_3]$
2. For $\lceil bnd/j \rceil$ batches: in parallel
    1. Evaluate j images of the inputs into new space in $\mathbb{Z}_p[x_1, x_2]$
    2. Make j recursive calls to $\textsc{MGCD}$ in parallel to get $G_i, \bar{A}_i, \bar{B}_i$

Parallel experiments in Cilk C

Parallel Implementation

Example: Call $\textsc{MGCD}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

1. Allocate space for interpolants $G^*, \bar{A}^*, \bar{B}^*$ in $\mathbb{Z}_p[x_1, x_2, x_3]$
2. For $\lceil bnd/j \rceil$ batches: in parallel
    1. Evaluate j images of the inputs into new space in $\mathbb{Z}_p[x_1, x_2]$
    2. Make j recursive calls to $\textsc{MGCD}$ in parallel to get $G_i, \bar{A}_i, \bar{B}_i$
    3. Distribute image GCD and cofactor coefficients into $G^*, \bar{A}^*, \bar{B}^*$

Parallel Implementation

Example: Call $\text{MGCD}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

1. Allocate space for interpolants $G^*, \bar{A}^*, \bar{B}^*$ in $\mathbb{Z}_p[x_1, x_2, x_3]$
2. For $\lceil bnd/j \rceil$ batches: in parallel
   1. Evaluate j images of the inputs into new space in $\mathbb{Z}_p[x_1, x_2]$
   2. Make j recursive calls to MGCD in parallel to get $G_i, \bar{A}_i, \bar{B}_i$
   3. Distribute image GCD and cofactor coefficients into $G^*, \bar{A}^*, \bar{B}^*$
3. interpolate $G^*, \bar{A}^*, \bar{B}^*$ in the univariate leaves in parallel

Parallel Implementation

Example: Call $\text{MGCD}(A, B)$ in $\mathbb{Z}_p[x_1, x_2, x_3]$

1. Allocate space for interpolants $G^*, \bar{A}^*, \bar{B}^*$ in $\mathbb{Z}_p[x_1, x_2, x_3]$
2. For $\lceil bnd/j \rceil$ batches: in parallel
   1. Evaluate j images of the inputs into new space in $\mathbb{Z}_p[x_1, x_2]$
   2. Make j recursive calls to MGCD in parallel to get $G_i, \bar{A}_i, \bar{B}_i$
   3. Distribute image GCD and cofactor coefficients into $G^*, \bar{A}^*, \bar{B}^*$
3. interpolate $G^*, \bar{A}^*, \bar{B}^*$ in the univariate leaves in parallel

The algorithm is recursive and needs a lot of pieces of memory.
Many calls to `malloc` can be a bad idea.
We allocate large blocks of memory and use it as a stack.
Memory for each bivariate Gcd is all preallocated.

Benchmarks $A, B \in \mathbb{Z}_p[x_1, x_2, x_3]$, $\deg A = \deg B = 200$.

Table : Real times in seconds, $p = 2^{62} - 57$, inputs have 1373701 terms

$\deg(G)$	$\deg(\bar{A})$	−opt,fft	−fft	1	8	16	20	Conv
10	190	15.81	8.79	4.79	0.84	0.54	0.48	0.37
40	160	14.59	9.42	5.79	0.92	0.55	0.49	0.27
70	130	13.25	9.74	6.47	0.99	0.56	0.49	0.21
100	100	11.80	9.87	6.72	1.00	0.57	0.50	0.18
130	70	10.25	8.19	5.29	0.80	0.46	0.40	0.18
160	40	8.56	7.14	4.16	0.66	0.39	0.34	0.20
190	10	6.80	6.58	3.44	0.58	0.37	0.33	0.25

jude 2 x E5-2680 v2 CPUs, 10 cores, 2.8 GHz (3.6 GHz turbo).
Maximum theoretical speed-up on 20 cores: 15.56

Benchmarks $A, B \in \mathbb{Z}_p[x_1, x_2, x_3]$, $\deg A = \deg B = 200$.

Table : Real times in seconds, $p = 2^{62} - 57$, inputs have 1373701 terms

Deg		Maple		MagmaR		MGCD, #CPUs				POLY
G	$\bar{A}$	$A \times B$	GCD	$A \times B$	GCD	1	4	8	16	Conv
10	190	2.22	70.98	77.22	33.34	6.35	1.83	1.06	0.71	0.47
40	160	25.65	267.16	920.48	159.71	7.75	2.13	1.18	0.75	0.35
70	130	25.62	439.80	1624.6	462.09	8.72	2.35	1.27	0.75	0.28
100	100	25.43	453.27	1526.2	900.65	9.11	2.43	1.32	0.79	0.24
130	70	25.69	436.11	1559.2	14254.	7.11	1.92	1.04	0.62	0.23
160	40	25.44	282.04	934.45	7084.3	5.63	1.52	0.83	0.51	0.26
190	10	2.23	77.28	90.30	2229.8	4.69	1.29	0.74	0.47	0.32

gaby two E5-2660 CPUs, 8 cores at 2.2 GHz (3.0 GHz turbo).
Maximum theoretical speed-up on 16 cores: 11.73

Current work

Let $G = \sum_{i=0}^{dg} g_i(x_2, \ldots, x_n) x_1^i$.
Let $t = \max_i \# g_i$.

Current work

Let $G = \sum_{i=0}^{dg} g_i(x_2, \ldots, x_n) x_1^i$.
Let $t = \max_i \#g_i$.

- Most of the time is evaluation: $O((\#A + \#B)t)$.

Current work

Let $G = \sum_{i=0}^{dg} g_i(x_2, \ldots, x_n) x_1^i$.
Let $t = \max_i \#g_i$.

- Most of the time is evaluation: $O((\#A + \#B)t)$.
- Have parallelized evaluation in batches of points.

Current work

Let $G = \sum_{i=0}^{dg} g_i(x_2, \ldots, x_n) x_1^i$.
Let $t = \max_i \#g_i$.

- Most of the time is evaluation: $O((\#A + \#B)t)$.
- Have parallelized evaluation in batches of points.
- Have parallelized on i sparse interpolation of $g_i(x_2, \ldots, x_n)$.

Current work

Let $G = \sum_{i=0}^{dg} g_i(x_2, \ldots, x_n) x_1^i$.
Let $t = \max_i \#g_i$.

- Most of the time is evaluation: $O((\#A + \#B)t)$.
- Have parallelized evaluation in batches of points.
- Have parallelized on i sparse interpolation of $g_i(x_2, \ldots, x_n)$.
- Need to switch to bivariate images.

Current work

Let $G = \sum_{i=0}^{dg} g_i(x_2, \ldots, x_n) x_1^i$.
Let $t = \max_i \#g_i$.

- Most of the time is evaluation: $O((\#A + \#B)t)$.
- Have parallelized evaluation in batches of points.
- Have parallelized on i sparse interpolation of $g_i(x_2, \ldots, x_n)$.
- Need to switch to bivariate images.