Fast parallel multi-point evaluation of sparse polynomials

Michael Monagan and Alan Wong. Department of Mathematics, Simon Fraser University

mmonagan@cecm.sfu.ca and cawong@sfu.ca

Introduction

The sparse polynomial GCD algorithm of Hu and Monagan [3] requires evaluating a multivariate polynomial A (with s terms) into t bivariate images, for some unknown $t \ll s$. These evaluations are the bottleneck of their algorithm, and our problem is to improve this. We outline their method below.

Input: $A = \sum_{i=1}^{s} a_i M_i(x_0, x_1, \ldots, x_n)$, $a_i \in \mathbb{Z}_p$

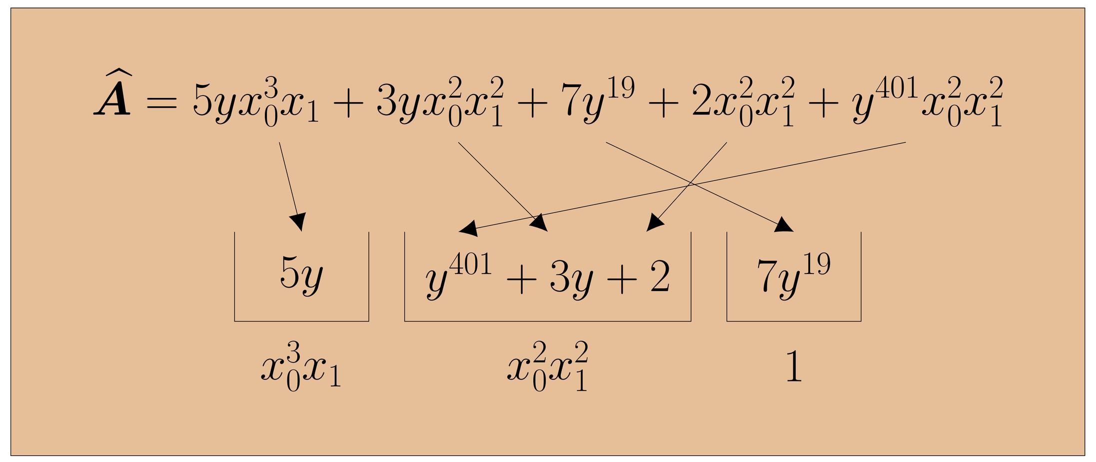
1. Kronecker map $A(x_0, x_1, \ldots, x_n) \mapsto \widehat{A}(x_0, x_1, y)$.
2. Let $\widehat{A}(x_0, x_1, y) = \sum_{i=1}^{s} a_i X_i y^{m_i}$, where X_i is a monomial in x_0, x_1. Find a primitive $\alpha \in \mathbb{Z}_p$ and compute $\beta_i = \alpha^{m_i}$ for $i = 1..s$.
3. Let T be the current guess for t. Evaluate $\widehat{A}$ at $y = \alpha^0, \alpha^1, \ldots, \alpha^{T-1}$ by computing $\gamma_i = \widehat{A}(x_0, x_1, \alpha^i)$ in the following matrix-vector multiplication:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \beta_1 & \beta_2 & \cdots & \beta_s \\ \vdots & \vdots & & \vdots \\ \beta_1^{T-1} & \beta_2^{T-1} & \cdots & \beta_s^{T-1} \end{bmatrix} \begin{bmatrix} a_1 X_1 \\ a_2 X_2 \\ \vdots \\ a_s X_s \end{bmatrix} = \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_{T-1} \end{bmatrix}$$

The above can be done in $O(sT + nd + ns)$ multiplications in $\mathbb{Z}_p$ [3]. Using the fast sparse multi-point evaluation described by van der Hoeven and Lecerf in [2] (originating from [1]), we can do better!

Our parallel algorithm and implementation reduces the $O(sT)$ cost to $O(s\log^2 T)$ under reasonable assumptions.

We begin by sorting the terms of $\widehat{A}$ into **buckets** on the monomials $x_0^j x_1^k$. Example:



We operate on each bucket separately as a sparse univariate polynomial in y.

Fast Sparse Multi-Point Evaluation

Let $\widehat{A}_{jk}(y) = \sum_{i=1}^{s_{jk}} a_i y^{m_i}$ be the polynomial in bucket $x_0^j x_1^k$. We parallelize on $\widehat{A}_{jk}$. The **main idea** of fast evaluation [2, 1]:

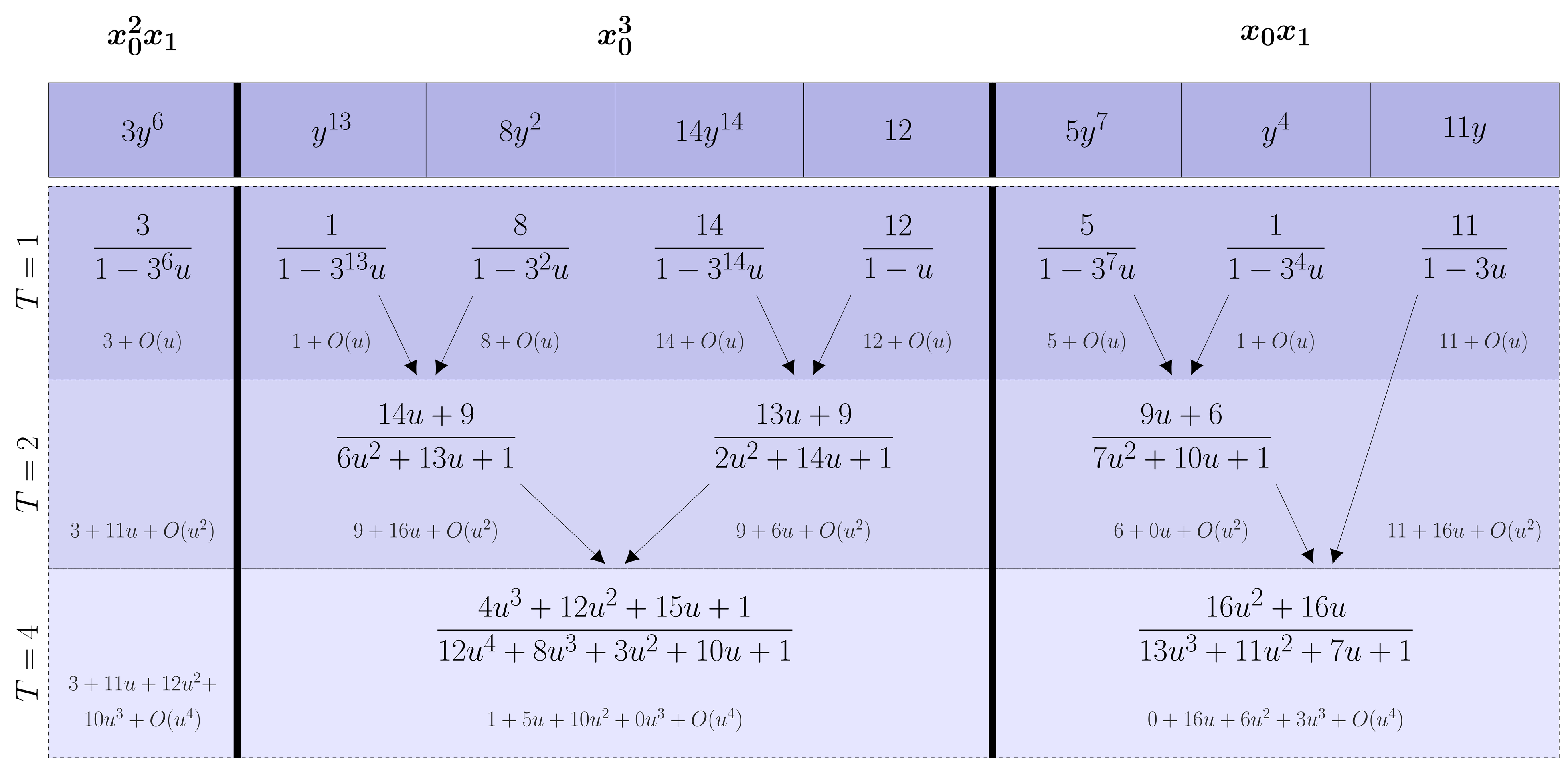
> $\widehat{A}_{jk}(\alpha^0), \ldots, \widehat{A}_{jk}(\alpha^{T-1})$ are the first T coefficients of the power series expansion of the rational function
>
> $$f(u) = \sum_{i=1}^{s_{jk}} \frac{a_i}{1 - \beta_i u}$$

- split $f(u)$ into blocks $B_1(u), \ldots, B_{\lceil s_{jk}/T \rceil}(u)$ of size $\le T$
- divide-and-conquer to compute the numerator/denominator of $B_i(u) = N_i(u)/D_i(u)$
- fast series inversion to get the power series expansion of $B_i(u)$ to $O(u^T)$
- cost: $O(\lceil \frac{s}{T} \rceil M(T) \log T) \to O(s \log^2 T)$ with FFT multiplication

As we don't know t, we use a bottom-up approach. Starting with a small guess T, we compute T evaluations to test for stabilization of the image GCD. If not stabilized, set $T := 2T$ and repeat. To combine two adjacent blocks of size T into a $2T$ block we use:

$$B_L + B_R = \underbrace{\frac{N_L}{D_L} + \frac{N_R}{D_R}}_{\text{from prev step}} = \underbrace{\frac{N_L D_R + N_R D_L}{D_L D_R}}_{\text{use fast multiplication}} = \frac{N}{D} \qquad (1)$$

We illustrate an example of the computation for $\widehat{A} = (3y^6)x_0^2x_1 + (y^{13} + 8y^2 + 14y^{14} + 12)x_0^3 + (5y^7 + y^4 + 11y)x_0x_1$ over $\mathbb{Z}_{17}$, with $\alpha = 3$:



Parallelize each level for N cores (using Cilk C):

- Count the number of total blocks b_T which require computing their N/D using (1). In the example $b_1 = 8$, $b_2 = 3$ and $b_4 = 2$. Assign $\lceil \frac{b_T}{N} \rceil$ blocks to each core.
- For the series expansion, we divide the buckets into N subsets of roughly equal work.

Benchmarks

- Generating random sparse polynomials with s terms and 9 variables
- degree in each variable ≤ 10, total degree ≤ 60
- run the algorithm until we get at least t images
- using an Intel Xeon server at 2.8/3.6 GHz, max theoretical speedup is **12.44** = 2.8/3.6 × 16

		Matrix			Fast		
s	t	1 core	16 cores	Speedup	1 core	16 cores	Speedup
10^7	10^2	7.35	0.73	10.0x	11.18	1.45	7.7x
10^7	500	32.67	2.71	12.0x	27.83	2.77	10.1x
10^7	10^3	64.32	5.29	12.2x	38.94	3.63	10.7x
10^7	10^4	633.51	51.43	12.3x	92.25	7.77	11.9x
10^7	10^5	6335.26	516.44	12.3x	155.58	12.72	12.2x
10^8	10^4	6198.68	553.84	11.2x	890.20	74.48	12.0x
10^8	10^5	-	5852.47	-	1374.74	112.52	12.2x
10^8	10^6	-	-	-	2045.96	164.96	12.4x

We inserted our fast evaluation implementation into the GCD code of [3]. Polynomials $G, \bar{A}, \bar{B}$ were created with $\#G, \#\bar{A}, \#\bar{B}$ terms (respectively), 9 variables, degree in each variable ≤ 20, and total degree ≤ 60.

We then constructed $A = G \cdot \bar{A}$ and $B = G \cdot \bar{B}$ as inputs to the GCD algorithm. t is the number of images required, and (eval) is the % of time spent in the evaluations. 16 cores were used for the Fast/Matrix timings.

$\#A$	$\#G$	t	Fast (eval)	Matrix (eval)	Maple	Magma
10^5	10^3	36	0.1 (76%)	0.1 (55%)	341.9	63.6
10^6	10^3	40	0.5 (88%)	0.2 (66%)	5553.5	FAIL
10^6	10^4	264	0.8 (82%)	0.6 (74%)	62520.1	FAIL
10^7	10^4	256	5.8 (90%)	4.5 (88%)	-	-
10^7	10^5	2334	13.5 (77%)	36.1 (91%)	-	-
10^7	10^6	24214	91.1 (32%)	395.7 (85%)	-	-
10^8	10^4	246	46.2 (89%)	45.8 (91%)	-	-
10^8	10^5	2328	96.3 (92%)	369.2 (98%)	-	-
10^8	10^6	24214	214.9 (69%)	3691.1 (98%)	-	-
10^8	10^7	242574	3058.1 (11%)	39643.0 (93%)	-	-

References

[1] A. Bostan, G. Lecerf, É. Schost. Tellegen's principle into practice. *Proceedings of ISSAC 2003*, ACM, 37–44, 2013.

[2] Joris van der Hoeven and Grégoire Lecerf. On the bit-complexity of sparse polynomial and series multiplication. *J. Symbolic Comput.*, **9**:227–254, 2013.

[3] Jiaxiong Hu and Michael Monagan. A fast sparse parallel polynomial GCD algorithm. Accepted for *ISSAC 2016*, 2016.

Mathematics • Modeling • Simulation