Implementing Kaltofen and Yagati's fast transposed Vandermonde solver

Hyukho Kwon and Michael Monagan
Department of Mathematics, Simon Fraser University
hyukhok@sfu.ca and mmonagan@sfu.ca

June 21, 2024

Extended Abstract

1 Introduction

We present a C implementation of Kaltofen and Yagati's fast transposed Vandermonde solver from [6] over the finite field $\mathbb{Z}_p$ for p a prime with at most 63 bits. For comparison, we have also implemented Kaltofen and Yagati's algorithm in Maple and Zippel's algorithm from [9] in C. The motivation for our work is the black-box multivariate polynomial factorization algorithm of Chen and Monagan [3] which needs to solve many transposed Vandermonde systems. In [3] the authors factor the determinant of the 16 by 16 symmetric Töplitz matrix. About 3/4 of the total factorization time is spent solving transposed Vandermonde systems, the largest of which is 127,690 by 127,690.

Let F be a field and $a(x) = \sum_{j=0}^{n-1} a_j x^j$ be an unknown polynomial in $F[x]$. For $u_1, u_2, \ldots, u_n \in F$, suppose we have computed $b_i = a(u_i)$ for $1 \le i \le n$ and we want to determine $a \in F^n$, that is, we want to interpolate $a(x)$. Various algorithms, for example [1], use geometric point sequences $u_i = \alpha^{i-1}$ for some $\alpha \in F$ such that $\alpha^i \neq \alpha^j$ for all $i \neq j$. Thus

$$b_i = a(u_i) = \sum_{j=0}^{n-1} a_j(\alpha^{i-1})^j = \sum_{j=0}^{n-1} a_j(\alpha^j)^{i-1} = \sum_{j=0}^{n-1} a_j(u_{j+1})^{i-1} \text{ for } 1 \le i \le n.$$

In matrix-vector form, we have

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ u_1 & u_2 & u_3 & \cdots & u_n \\ u_1^2 & u_2^2 & u_3^2 & \cdots & u_n^2 \\ \vdots & \vdots & \vdots & & \vdots \\ u_1^{n-1} & u_2^{n-1} & u_3^{n-1} & \cdots & u_n^{n-1} \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}}_{a} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}}_{b}. \tag{1}$$

The matrix U is called a transposed Vandermonde matrix and the linear system $Ua = b$ is called a transposed Vandermonde system.

We note that in some applications, for example, the GCD algorithm of Hu and Monagan in [5], one needs random evaluation points for u_i as $u_1 = \alpha^0 = 1$ may cause a problem. To resolve

this we may instead use $u_i = \alpha^i$ for $1 \le i \le n$ so that $u_1 = \alpha$. This leads to a shifted transposed Vandermonde system $U'a = b$ where $U'_{i,j} = u_j^i$ for $1 \le i \le n$ and $1 \le j \le n$. The matrix U' factors as $U' = UD$ where D is a diagonal matrix with $D_{i,i} = u_i$. Thus to solve $U'a = (UD)a = b$, since $a = D^{-1}U^{-1}b$ we first solve the unshifted transposed Vandermonde system $Uc = b$ for c then use $a = D^{-1}c$.

Let $M(n)$ be the number of field operations for polynomial multiplication with two polynomials of degree at most n in $F[x]$. Table 1 summarizes three methods for solving $Ua = b$.

Methods	# ops in F	space
Gaussian Elimination	$O(n^3)$	$O(n^2)$
Zippel's method [9]	$O(n^2)$	$O(n)$
Kaltofen & Yagati's method [6]	$O(M(n)\log n)$	$O(n \log n)$

Table 1: Algorithms for solving n by n transposed Vandermonde systems

2 Summary of work and optimizations

Kaltofen and Yagati's algorithm assumes fast multiplication, fast multi-point evaluation and fast division in $F[x]$. We summarize what we have implemented for these for the prime field $F = \mathbb{Z}_p$.

2.1 Fast multiplication

Let q be a 63 bit Fourier prime, that is, a prime of the form $q = 2^k s + 1$ for large k. The underlying FFT for $\mathbb{Z}_q^n$ that we use is the in-place recursive FFT from Law and Monagan [7] which does exactly $\frac{1}{2} n \log_2 n$ multiplications. For ω a primitive nth root of unity in $\mathbb{Z}_q$ it precomputes an array W of size n of the powers of ω needed for all recursive calls

$$W = \boxed{1}\boxed{\omega}\boxed{\omega^2}\boxed{\cdots}\boxed{\omega^{n/2-1}}\boxed{1}\boxed{\omega^2}\boxed{\omega^4}\boxed{\cdots}\boxed{\omega^{n/2-2}}\boxed{\cdots}\boxed{1}\boxed{0}$$

To multiply two polynomials in $\mathbb{Z}_q[x]$, Law and Monagan use the decimation in frequency FFT for the two forward transforms and the decimation in time FFT for the inverse transform so that the two bit-reversal permutations cancel out and can be omitted from both FFTs.

Let $f, g \in \mathbb{Z}_p[x]$. To multiply modulo p we choose three Fourier primes p_1, p_2, p_3 such that $p_1 p_2 p_3 > (p-1)^2 \min(1 + \deg(f), 1 + \deg(g))$ and we use the FFT to multiply $f \times g \mod p_i$ for $i = 1, 2, 3$ then Chinese remaindering to recover the integer coefficients in $f \times g$ before reduction mod p. This is called the three primes method. In [8], von zur Gathan and Gerhard use it for multiplying long integers.

Theorem 1 *The three primes method does $M(n) = 27n \log_2 n + O(n)$ field operations to multiply two polynomials of degree at most n in $\mathbb{Z}_p[x]$.*

2.2 Fast division

Let $f, g \in \mathbb{Z}_p[x]$ with $\deg(g) = n$ and $\deg(f) < 2n$. The classical algorithm for f divided by g does $O(n^2)$ field operations. Let $g = \sum_{i=0}^{n} g_i x^i$ and $\widehat{g} = \sum_{i=0}^{n} g_{n-i} x^i$ denote the reciprocal polynomial. The fast division algorithm [8] computes $\widehat{g}^{-1} \mod x^n$ using a Newton iteration (see Theorem 2 below) then the quotient q from $\widehat{q} = \widehat{f} \times \widehat{g}^{-1} \mod x^n$ then the remainder r using $r = f - g \times q$.

Theorem 2 *(Theorem 9.2 [8]) Assume* $h = \sum_{i=0}^{n} h_i x^i \in \mathbb{Z}_p[x]$ *and* $h_0 \neq 0$. *Let* $y_0 = h_0^{-1}$ *and* $y_i = 2y_{i-1} - h \cdot y_{i-1}^2 \mod x^{2^i}$ *for* $i \geq 1$. *For all* $i \geq 0$, $h \cdot y_i \mod x^{2^i} = 1$.

Using Theorem 2, we can compute the inverse of $\hat{g}$ in $3M(n) + O(n)$ field operations in $\mathbb{Z}_p$ [8]. We have implemented the middle product of Hanrot, Quercia, and Zimmerman [4] who instead use $y_i = y_{i-1} + y_{i-1} \cdot (1 - h \cdot y_{i-1})$. Their method reduces the cost of computing $\widehat{g}^{-1}$ to $2M(n) + O(n)$ field operations in $\mathbb{Z}_p$. All polynomial multiplications need to be done using the three primes method. Using the three primes method, we can also save one FFT by computing the FFT of y_{k-1} once which reduces the constant 2 to 5/3. The total cost for the polynomial multiplications in division becomes $11/3M(n) + O(n)$ field operations in $\mathbb{Z}_{p_i}$.

Theorem 3 *Fast division in* $\mathbb{Z}_p[x]$ *does at most* $11M(n) + O(n)$ *field operations.*

In our implementation of fast division we use classical division for $n \leq 512$.

2.3 Fast multi-point evaluation

Let f be a polynomial in $\mathbb{Z}_p[x]$ with $\deg(f) \leq n-1$ where $n = 2^k$ for some $k \geq 0$. Let $u_0, u_1, \ldots, u_{n-1}$ be distinct elements in $\mathbb{Z}_p$. The multi-point evaluation problem is to compute $f(u_i)$ for $0 \leq i \leq n-1$.

Repeated usage of Horner's method costs $O(n^2)$ arithmetic operations in $\mathbb{Z}_p$. In 1971, Borodin and Munro [2] introduced an $O(M(n) \log n)$ algorithm. Their algorithm first builds a product tree T, a complete binary tree in which every leaf is a linear polynomial $x - u_i$ for $0 \leq i \leq n-1$ and each parent node is the product of their two children so that the root node of T is $T_{k,0} = \prod_{i=0}^{n-1} x - u_i$.

Each multiplication in the product tree is of two monic polynomials $T_{i,j} = x^{2^i} + A(x)$ by $T_{i,j+1} = x^{2^i} + B(x)$ which needs an FFT of size 4×2^i. Instead, by computing $T_{i,j} \times B(x)$ and adding $x^{2^i} T_{i,j}$ we can use an FFT of size 2×2^i. Also, since all polynomials in T are monic we only need to store $A(x)$ and $B(x)$ which have size 2^i. The product tree can be stored in a one dimensional array using space for $n(1 + \log_2 n)$ elements of $\mathbb{Z}_p$.

Theorem 4 *Building a product tree (BuPT) in* $\mathbb{Z}_p[x]$ *does* $\frac{3}{2}M(n) \log_2 n + O(n \log n)$ *field operations.*

Let $m_i = x - u_i$ for all $0 \leq i \leq n-1$. The remainder of $f(x)$ divided m_i, denoted $f \mod m_i$ is $f(u_i)$. Recall that if $g|h$, then $f \mod g = (f \mod h) \mod g$ where $f, g, h \in \mathbb{Z}_p[x]$. Each node $T_{i,j}$ for $0 \leq i \leq k, 0 \leq j < 2^{k-i}$ in the product tree T is a factor of its parent node in T. In other words,

$$T_{i,j} = T_{i-1,2j} \times T_{i-1,2j+1} \implies T_{i-1,2j} | T_{i,j} \text{ and } T_{i-1,2j+1} | T_{i,j}.$$

Thus we can compute $f(u_i)$ for $0 \leq i \leq n-1$ by dividing down the product tree with a divide-and-conquer approach.

Theorem 5 *Dividing down the product tree (DDPT) in* $\mathbb{Z}_p[x]$ *does* $11M(n) \log_2 n + O(n \log n)$ *field operations.*

2.4 Fast transposed Vandermonde solver

We present Kaltofen and Yagati's algorithm from [6] in Algorithm 1. Given $u_1, u_2, \ldots, u_n$, Kaltofen and Yagati first build the product tree T. After constructing T, we extract the root polynomial M, which Zippel [9] calls the master polynomial. Then Algorithm 1 creates a polynomial D whose coefficients are from b and multiplies M by D to get $H = M \times D$. We construct a polynomial Q from H. The dominating cost is evaluating Q and M' at $u_1, u_2, \ldots, u_n$ by dividing down the product tree T to obtain $a_{i-1} = Q(u_i)/M'(u_i)$ for $1 \leq i \leq n$.

Algorithm 1 Fast transposed Vandermonde solver(FastTVS)

Input: $n = 2^k$ for some $k \in \mathbb{N}$, $u = [u_1, u_2, \ldots, u_n] \in \mathbb{Z}_p^n$ (which defines the transposed Vandermonde matrix U) and $b = [b_1, b_2, \ldots, b_n] \in \mathbb{Z}_p^n$

Output: $a = [a_0, a_1, \ldots, a_{n-1}] \in \mathbb{Z}_p^n$ satisfying $Ua = b$.

1: $T \leftarrow \text{BuPT}(n, u)$ ..... $\frac{3}{2}M(n)\log_2 n + O(n \log n)$
2: $M \leftarrow T_{k,0}$ from T // $M = \prod_{i=1}^{n}(x - u_i)$
3: $D \leftarrow b_n x + b_{n-1}x^2 + \cdots + b_1 x^n$
4: $H \leftarrow M \times D$ // Let $H = \sum_{i=0}^{2n-1} h_i x^{i+1}$ ..... $3M(n)$
5: $Q \leftarrow \sum_{i=0}^{n-1} h_{n+i}x^i$ // $\sum_{i=0}^{n-1} h_{n+i}z^i$ is the coefficient of x^n in $H/(x - z)$
6: $s_1, s_2, \ldots, s_n \leftarrow \text{DDPT}(n, Q, T)$ // $s_i = Q(u_i)$ ..... $11M(n)\log_2 n + O(n \log n)$
7: Differentiate M ..... $O(n)$
8: $t_1, t_2, \ldots, t_n \leftarrow \text{DDPT}(n, M', T)$ // $t_i = M'(u_i) = q_i(u_i)$ ..... $11M(n)\log_2 n + O(n \log n)$
9: **for** i from 1 to n **do** $a_{i-1} \leftarrow t_i^{-1} \cdot s_i$ **end for** ..... $O(n)$
10: **return** $[a_0, a_1, \ldots, a_{n-1}]$

Theorem 6 *Algorithm 1 does at most* $\frac{53}{2}M(n)\log_2 n + O(n \log n)$ *field operations in* $\mathbb{Z}_p$.

In our implementation of Algorithm 1, for $n \leq 64$ we evaluate using Horner's method instead of dividing down the product tree. Also, if after Step 1 we compute the inverses of all $\widehat{T}_{i,j}$ polynomials, we can use them for both DDPT calls in Steps 6 and 8. Computing intermediate inverses costs $5M(n)\log_2 n + O(n \log n)$. This reduces the cost of each DDPT to $6M(n)\log_2 n + O(n \log n)$ field operations. In our implementation we store the inverses in a second product tree. This optimization reduced the time for $n = 2^{16}$ in Table 2 from 1500.7 ms to 1249.3 ms.

2.5 Implementation and Benchmark

We have implemented Algorithm 1 FastTVS in C for the case where p is a 63 bit Fourier prime and are presently implementing the three primes method.

For comparison we have also implemented Zippel's $O(n^2)$ algorithm from [9] in C and we have implemented Kaltofen and Yagati's algorithm in Maple. The Maple implementation for multiplication in $\mathbb{Z}_p[x]$ is done using a single large integer multiplication using GMP's fast integer multiplication.

The timings in Table 2 are in milliseconds. They were obtained on an AMD FX 8350-8 8-core CPU at 4.2GHz using one core. The ratios in the first speed up column are the timings in column ZippelTVS divided by those in column Total. The ratios in the second speed up column are the timings in column Maple divided by those in column Total.

Our FastTVS implementation beats Zippel's $O(n^2)$ method for $n > 128$ which is a good result. Notice also that the time to build the product tree (BuPT) is much smaller than the time to compute the inverses (column InvTree) plus divide down the product tree (columns DDPT1 and DDPT2) thus any improvement will need to focus on polynomial division. Also, the Maple time of 69,705 ms for $n = 2^{18}$ is not an error; it seems to be an anomaly.

Acknowledgement

This work was supported by NSERC of Canada and Maplesoft.

n	FastTVS					ZippelTVS	speed up	Maple	speed up
	BuPT	InvTree	DDPT1	DDPT2	Total				
2^6	0.046	-	0.046	0.039	0.195	0.1389	0.71	3.4	17.4
2^7	0.086	-	0.107	0.098	0.380	0.4879	1.28	8.6	22.6
2^8	0.150	-	0.254	0.238	0.808	1.9039	2.35	20.8	25.7
2^9	0.363	-	0.693	0.674	2.065	7.4640	3.61	63.0	30.5
2^{10}	0.875	0.600	1.890	1.877	5.811	30.826	5.30	113.2	19.5
2^{11}	2.020	2.417	5.070	5.008	15.775	116.84	7.40	270.0	17.1
2^{12}	4.755	7.529	12.307	12.268	39.444	469.64	11.90	608.0	15.4
2^{13}	11.146	20.556	29.566	29.270	95.765	1,868	19.50	1,321	13.8
2^{14}	25.901	53.099	71.091	70.580	231.55	7,456	32.19	3,025	13.1
2^{15}	60.151	131.30	166.15	166.46	546.52	29,986	54.86	7,190	13.2
2^{16}	131.23	314.56	380.02	376.77	1,249.3	120,292	96.28	16,455	13.2
2^{17}	339.89	746.70	867.30	863.48	2,914.9	478,912	164.3	69,705	23.9
2^{18}	663.01	1,747.1	1,961.8	1,955.2	6,529.8	1,929,776	295.5	97,667	15.0

Table 2: CPU timings in ms for solving $n \times n$ transposed Vandermonde systems over the prime field $\mathbb{Z}_p$ with $p = 116 \cdot 2^{55} + 1$

References

[1] Michael Ben-Or and Prasoon Tiwari: A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation. *Proceedings of STOC '20*, pp. 301–309, ACM, 1988.

[2] A. Borodin and I. Munro: Evaluating polynomials at many points. *Information Processing Letters* **1**(2):66–68, 1971.

[3] Tian Chen and Michael Monagan: Factoring Multivariate Polynomials Represented by Black Boxes – A Maple + C Implementation. *Mathematics in Computer Science* **16**(2–3), article 18, Springer, 2022. https://doi.org/10.1007/s11786-022-00534-7

[4] Guillaume Hanrot, Michel Quercia, Paul Zimmermann: The Middle Product Algorithm I. Speeding up the division and square root of power series. *Applicable Algebra in Engineering, Communication and Computing* **14**(6):415–438, Springer, 2004.

[5] Jiaxong Hu and Michael Monagan: A fast parallel sparse polynomial GCD algorithm. *Proceedings of ISSAC '2016*, pp. 271–278, ACM, 2016.

[6] Erich Kaltofen and Lakshman Yagati: Improved sparse multivariate polynomial interpolation algorithms. *Proceedings of ISSAC'88,* pp. 467–474, Springer, 1988.

[7] Marshall Law and Michael Monagan: A parallel implementation for polynomial multiplication modulo a prime. *Proceedings of PASCO'2015,* pp. 78–86, ACM, 2015.

[8] Joachim von zur Gathen and Jüergen Gerhard: *Modern Computer Algebra,* Cambridge University Press, 2013.

[9] Richard Zippel: Interpolating polynomials from their values. *Journal of Symbolic Computation* **9**(3):375–403, Elsevier, 1990.