A Maple implementation of FFT-based algorithms for polynomial multipoint evaluation, interpolation, and solving transposed Vandermonde systems

Kimberly Connolly

Dept. of Mathematics, Simon Fraser University, British Columbia

Three Main Problems

Let $f(x) = a_0 + a_1 x + ... + a_{n-2} x^{n-2} a_{n-1} x^{n-1} \in F[x]$.

Multipoint evaluation: Given n arbitrary points, $u_0, u_1, ..., u_{n-1}$, compute $f(u_i) = v_i$ for $0 \leq i < n$.

Interpolation: Given $f(u_i) = v_i$ for $0 \leq i < n$, reconstruct the polynomial f.

Transposed Vandermonde systems: Let V be a Vandermonde matrix of order n, solve $V^T b = v$ for b.

Classically, algorithms for the above three problems have an $O(n^2)$ running time. How can we improve this to $O(n \log^2 n)$?

Approach: All three fast algorithms assume fast multiplication and fast division in $F[x]$ and make use of a **suproduct tree**.

For fast multiplication in $\mathbb{F}_p[x]$, we use the FFT.
For fast division in $\mathbb{F}_p[x]$, we use Newton iteration.

Talk Outline

1. The Fast Fourier Transform (FFT)
2. Fast polynomial multiplication in $\mathbb{F}_p[x]$ using the FFT
3. Table of complexities
4. The subproduct tree
5. Fast multipoint evaluation
6. Fast interpolation
7. Fast transposed Vandermonde systems
8. Summary and future work

The Fast Fourier Transform (FFT)

Definition

Let $w \in F$ be a primitive nth root of unity, then the Discrete Fourier Transform (DFT) calculates $[f(1), f(w), f(w^2), ..., f(w^{n-1})] \in F^n$. The FFT is an algorithm that computes the DFT in $O(n \log n)$ arithmetic operations in F.

Input: $n = 2^k, a = [a_0, a_1, ..., a_{n-1}] \in \mathbb{F}_p^n$, p is prime, $w \in \mathbb{F}_p$ of order n, and $W = [1, w, w^2, ..., w^{n/2-1}] \in \mathbb{F}_p^{n/2}$.

Output: $A = [f(1), f(w), f(w^2), ..., f(w^{n-1})] \in \mathbb{F}_p^n$, where $f(x) = \sum_{i=0}^{n-1} a_i x^i$.

Let $\mathbf{F_n}$ be the number of arithmetic operations in F required to compute an FFT of size n. Then, recurrence relation for the FFT is

$$\mathbf{F_n} = 2\mathbf{F_{n/2}} + 3(n/2).$$

Theorem (1)

$$\mathbf{F_n} = \frac{3}{2}(n \log n)$$

Fast multiplication in $\mathbb{F}_p[x]$ using the FFT

Let $f(x) = \sum_{i=0}^{d} a_i x^i$, $g(x) = \sum_{i=0}^{d} b_i x^i$, and $h(x) = f(x) \times g(x) \in \mathbb{F}_p[x]$.

Input: $n = 2^k, A = [a_0, a_1, ..., a_d, 0, ..., 0] \in \mathbb{F}_p^n, B = [b_0, b_1, ..., b_d, 0, ..., 0] \in \mathbb{F}_p^n$, and $w \in \mathbb{F}_p$ of order n, where p is a prime.

Output: $c = [c_0, c_1, ..., c_{2d}, 0, ..., 0] \in \mathbb{F}_p^n$.

0 Compute $W = [1, w, ..., w^{\frac{n}{2}-1}]$ and $W' = [1, w^{-1}, ..., w^{1-\frac{n}{2}}]$. . . . . $n+1$.

1 $a \leftarrow \text{FFT}(n, A, w, W, p)$. . . . . $\mathbf{F_n}$.

2 $b \leftarrow \text{FFT}(n, B, w, W, p)$. . . . . $\mathbf{F_n}$.

3 $C \leftarrow [a_0 \cdot b_0, a_1 \cdot b_1, ..., a_{n-1} \cdot b_{n-1}]$ . . . . . n.

4 $c \leftarrow \text{FFT}(n, C, w^{-1}, W', p)$ . . . . . $\mathbf{F_n}$.

5 $c \leftarrow n^{-1} \cdot [c_0, c_1, ..., c_{n-1}]$ . . . . . $n+1$.

Theorem (2)

Let $M(n)$ be the number of arithmetic operations in F required for a fast mutliplication of two polynomials of degree less than or equal to n. Then,

$$M(n) = 3\mathbf{F_n} + 3n + 2 = \frac{9}{2}(n \log n) + O(n) \in O(n \log n).$$

Table of complexities

$M(n)$ is the cost of multiplying two polynomials of degree at most n.

Problem	**Classic**	**Fast Method with $M(n)$**	**Fast Method with FFT**
Division	$O(n^2)$	$4M(n) + O(n)$	$O(n \log n)$
Evaluation	$O(n^2)$	$O(M(n) \log n)$	$O(n \log^2 n)$
Interpolation	$O(n^2)$	$O(M(n) \log n)$	$O(n \log^2 n)$
Vandermonde	$O(n^2)$	$O(M(n) \log n)$	$O(n \log^2 n)$

Table: Comparing the complexities for fast division, fast mutlipoint evaluation, fast interpolation, and fast solving transposed Vandermonde systems

Let the evaluation points be: $u_0 = 4, u_1 = 3, u_2 = 2$, and $u_3 = 1$. Then,

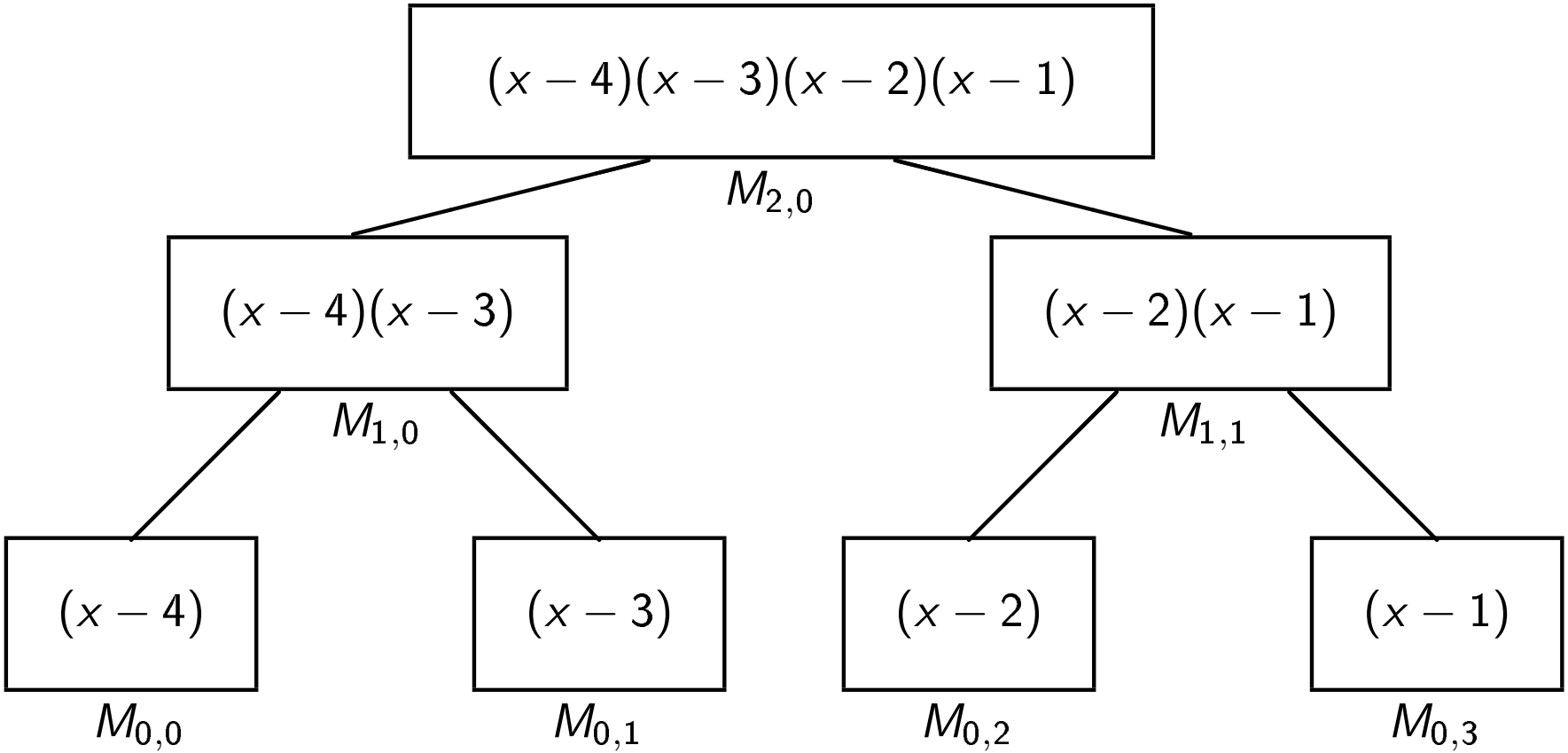

Figure: Example of Subproduct Tree when $n = 4$

- The subproduct tree has height $k = \log_2 n$.
- The n leaves are the linear polynomials $M_{0,j} = x - u_i$ for $0 \leq i < n$.
- The root is defined by the largest polynomial $M_{k,0} = \prod_{i=0}^{n-1}(x - u_i)$.
- $M_{i,j} = M_{i-1,2j} \times M_{i-1,2j+1}$.

Lemma for Complexity Analysis

Lemma (1)

Let $n = 2^k$ and a, b, d be natural numbers with $b > 0$ and let c be a positive real number. Let S and T be functions where $S(n/2) = c \cdot (n \log_2 n)$, and

$$T(1) = a, \quad T(n) \leq 2T(n/2) + bS(n/2) + dn$$

Then, we have:

$$T(n) < b \cdot \frac{1}{4} S(n) \log n + b \cdot \frac{1}{4} S(n) + d(n \log n) + na$$

Complexity of Subproduct Tree Algorithm (BUST)

Let $B(n)$ be the number of arithmetic operations in F needed to compute the subproduct tree. Then,

$$B(n) = 2B(n/2) + 1M(n/2) + 0 \cdot n$$

For $n = 1$, we must return $x - u_0$ and so $B(1) = 1$. Let $M(n/2) = c \cdot (n \log_2 n)$, so we can use Lemma 1. In this case, $a = 1$, $b = 1$, and $d = 0$ and we find that:

$$\begin{aligned} B(n) &< 1 \cdot \frac{1}{4}M(n)\log n + 1 \cdot \frac{1}{4}M(n) + 0 \cdot (n \log n) + n \cdot 1 \\ &= \frac{1}{4}M(n)\log n + \frac{1}{4}M(n) + O(n) \end{aligned}$$

The complexity analysis of the subtree algorithm is summarized in Theorem 3.

Theorem (3)

$$B(n) < \frac{1}{4}M(n)\log n + \frac{1}{4}M(n) + O(n) \in O(n \log^2 n).$$

Dividing Down the Subproduct Tree (DDST)

Recall $f(u_i) = f \text{ rem } (x - u_i)$ for all $0 \leq i < n$. Now, to find $f(u_i) = v_i$ for $0 \leq i < n$, we will **recurse down the subtree**. Let $m_i = (x - u_i)$, we define:

$$r_0 = f \text{ rem } \prod_{i=0}^{n/2-1} m_i = f \text{ rem } M_{k-1,0} \text{ and } r_1 = f \text{ rem } \prod_{i=n/2}^{n-1} m_i = f \text{ rem } M_{k-1,1}.$$

1. If $n = 1$ then return f.
2. Compute $r_0 = f \text{ rem } M_{k-1,0}$ using the fast division algorithm. ..... $D(n/2)$.
3. Compute $r_1 = f \text{ rem } M_{k-1,1}$ using the fast division algorithm. ..... $D(n/2)$.
4. Call the algorithm recursively at the subtree rooted at $M_{k-1,0}$ to compute $r_0(u_0), ..., r_0(u_{n/2-1})$.
5. Call the algorithm recursively at the subtree rooted at $M_{k-1,1}$ to compute $r_1(u_{n/2}), ..., r_1(u_{n-1})$.
6. **return** $r_0(u_0), ..., r_0(u_{n/2-1}),\ r_1(u_{n/2}), ..., r_1(u_{n-1}) = f(u_0), ..., f(u_{n-1})$

Complexity of DDST and Fast Multipoint Evaluation

Let $C(n)$ be the number of arithmetic operations in F that DDST does. Then,

$$C(n) = 2C(n/2) + 2D(n/2) + 0 \cdot n$$

We have $C(1) = 0$. Using Lemma 1 with $a = 0$, $b = 2$, and $d = 0$.

Theorem (4)

$$C(n) < \frac{1}{2}D(n)\log\ n + \frac{1}{2}D(n) \in O(n\ \log^2\ n).$$

The fast multipoint evaluation algorithm simply calls BUST and DDST.

Let $E(n)$ be number of arithmetic operations in F needed to evaluate a degree $n-1$ polynomial f at n arbitrary points, $u_0, ..., u_{n-1}$, using the fast method.

$$E(n) = B(n) + C(n) < \frac{1}{4}M(n)\log\ n + \frac{1}{4}M(n) + O(n) + \frac{1}{2}D(n)\log\ n + \frac{1}{2}D(n)$$

Theorem (5)

$$E(n) < 11(n\ \log^2\ n) + O(n\ \log\ n) + O(n) \in O(n\ \log^2\ n).$$

Fast Multipoint Evaluation (FastEval) Timings

We implemented FastEval in Maple in $\mathbb{F}_p[x]$, with $p = 7 \cdot 2^{26} + 1$. For the quadratic timings, we used the `eval` command in Maple n times. The timings were run on an Intel Xeon E5 2660 CPU with 64 gigabytes of RAM.

n	Quadratic	Growth Factor	BUST	FastEval	Growth Factor
2^8	112 ms	N/A	7 ms	79 ms	N/A
2^9	430 ms	3.84	15 ms	141 ms	1.78
2^{10}	1.87 s	4.35	34 ms	296 ms	2.10
2^{11}	7.61 s	4.07	58 ms	607 ms	2.05
2^{12}	31.93 s	4.20	120 ms	1.32 s	2.18
2^{13}	2.01 min	3.78	227 ms	2.89 s	2.19
2^{14}	8.17 min	4.06	432 ms	6.14 s	2.12
2^{15}	33.51 min	4.10	917 ms	13.48 s	2.20
2^{16}	2.33 hr	4.17	1.92 s	29.26 s	2.17
2^{17}	9.83 hr	4.22	4.15 s	1.01 min	2.07
2^{18}	39.61 hr	4.03	8.42 s	2.07 min	2.06

Table: The timings of FastEval in Maple

Fast Interpolation (FastInterp)

The Lagrange interpolant

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n-1} \frac{x - u_j}{u_i - u_j} \text{ has the property } L_i(u_j) = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \end{cases}.$$

In Lagrange interpolation, the interpolating polynomial $f(x)$ takes the form

$$f(x) = \sum_{i=0}^{n-1} v_i L_i(x) = \sum_{i=0}^{n-1} v_i \prod_{\substack{j=0 \\ j \neq i}}^{n-1} \frac{x - u_j}{u_i - u_j} \tag{1}$$

$$= \sum_{i=0}^{n-1} v_i \frac{t_i M(x)}{x - u_i} \text{ where } M(x) = \prod_{j=0}^{n-1} x - u_j \text{ and } t_i = \prod_{j \neq i} \frac{1}{u_i - u_j}.$$

We will take $M'(x) = \sum_{0 \leq j < n} M(x)/(x - u_i)$, which is the formal derivative of $M(x)$, and note that $M(x)/(x - u_i)$ vanishes at all points u_j with $i \neq j$. Thus,

$$\frac{1}{t_i} = M'(u_i) = \frac{M}{x - u_i}\Big|_{x=u_i}$$

By Theorem 5, we can compute the t_i's in $O(n \log^2 n)$ arithmetic operations in F.

FastInterp Cont.

Let $n = 2^k$ and $c_i = v_i t_i$ for $0 \leq i < n$. Then, given the polynomials $M_{i,j}$ from the subtree, we can use a divide and conquer algorithm, called **InterpWork**, to recursively find:

$$r_0 = \sum_{i=0}^{n/2-1} c_i \frac{M_{k-1,0}}{x - u_i} \quad \text{and} \quad r_1 = \sum_{i=n/2}^{n-1} c_i \frac{M_{k-1,1}}{x - u_i}$$

Next, we obtain f using $f = M_{k-1,1} r_0 + M_{k-1,0} r_1$ and then we return f. Thus, we can implement fast interpolation with two algorithms. The first is **InterpWork** which is the core of fast interpolation and outputs

$$f = \sum_{i=0}^{n-1} c_i \frac{M(x)}{x - u_i} \in \mathbb{F}_p[x] \text{ where } M(x) = M_{k,0}.$$

The second, called **FastInterp**, pulls everything together by providing the input for InterpWork.

Complexity of FastInterp

Let $T(n)$ be the number of arithmetic operations in F done by InterpWork. Then,

$$T(n) \leq 2T(n/2) + 2M(n/2) + 1 \cdot n$$

If $n = 1$, we simply return c_0, and so $T(1) = 0$. Let $M(n/2) = c \cdot (n \log_2 n)$, so we can use Lemma 1 with $a = 0$, $b = 2$, and $d = 1$.

$$T(n) < \tfrac{1}{2}M(n)\log n + \tfrac{1}{2}M(n) + O(n \log n) \in O(n \log^2 n)$$

Let $I(n)$ be number of arithmetic operations in F needed to interpolate a polynomial of degree $n-1$, using the fast method. Then,

$$I(n) = B(n) + C(n) + T(n) + O(n).$$

Theorem (6)

$$I(n) < 13(n \log^2 n) + O(n \log n) + O(n) \in O(n \log^2 n).$$

FastInterp Timings

We implemented FastInterp in Maple in $\mathbb{F}_p[x]$, with $p = 7 \cdot 2^{26} + 1$. For the quadratic algorithm, the `Interp(...) mod p` command in Maple was used, which employs Newton interpolation. The timings were run on an Intel Xeon E5 2660 CPU with 64 gigabytes of RAM.

n	Quadratic	Growth	BUST	InterpWork	FastInterp	Growth
2^8	2 ms	N/A	6 ms	5 ms	85 ms	N/A
2^9	6 ms	3.00	11 ms	11 ms	131 ms	1.54
2^{10}	21 ms	3.50	21 ms	23 ms	255 ms	1.95
2^{11}	73 ms	3.48	43 ms	45 ms	658 ms	2.58
2^{12}	277 ms	3.79	78 ms	81 ms	1.21 s	1.84
2^{13}	1.09 s	3.94	154 ms	190 ms	2.44 s	2.02
2^{14}	4.32 s	3.96	332 ms	400 ms	5.07 s	2.07
2^{15}	17.50 s	4.05	754 ms	856 ms	11.04 s	2.17
2^{16}	1.19 min	4.08	1.51 s	1.83 s	23.01 s	2.08
2^{17}	4.79 min	4.03	3.49 s	3.92 s	50.76 s	2.21
2^{18}	19.24 min	4.02	7.61 s	8.39 s	1.74 min	2.06

Table: Timings of FastInterp in Maple

Complexity of FastVandermonde and Timings

Theorem (7)

Let $V(n)$ be number of arithmetic operations in F required to solve a transposed Vandermonde system, using the fast method. Then,

$$V(n) = B(n)+2C(n)+M(n)+O(n) < 20(n \log^2 n)+O(n \log n)+O(n) \in O(n \log^2 n).$$

We implemented FastVandermonde in Maple in $\mathbb{F}_p[x]$, with $p = 7 \cdot 2^{26} + 1$. For the quadratic timings, we coded Zippel's algorithm from 1990.

n	Quadratic	Growth Factor	BUST	FastVandermonde	Growth Factor
2^{11}	1.14 s	4.13	92 ms	1.38 s	1.99
2^{12}	4.78 s	4.19	182 ms	3.01 s	2.18
2^{13}	18.17 s	3.80	376 ms	6.09 s	2.02
2^{14}	1.32 min	4.34	841 ms	13.34 s	2.19
2^{15}	5.16 min	3.92	1.69 s	26.47 s	1.98
2^{16}	20.40 min	3.95	3.58 s	1.07 min	2.41
2^{17}	1.36 hr	4.01	7.45 s	1.96 min	1.83
2^{18}	5.55 hr	4.08	14.97 s	4.42 min	2.26

Table: Timings of FastVandermonde in Maple

Summary and Future Work

Classical Methods: $\mathbf{O(n^2)}$

FastEval:

$$E(n) < 11(n \log^2 n) + O(n \log n) + O(n) \in \mathbf{O(n \log^2 n)}.$$

FastInterp:

$$I(n) < 13(n \log^2 n) + O(n \log n) + O(n) \in \mathbf{O(n \log^2 n)}.$$

Fast Vandermonde:

$$V(n) < 20(n \log^2 n) + O(n \log n) + O(n) \in \mathbf{O(n \log^2 n)}.$$

Future Work:

- Code these algorithms in C and compare the timings to Maple.
- Implement a fast Chinese remaindering algorithm that uses the subproduct tree, but also requires a fast Euclidean algorithm.

References

A. Borodin and R. Moenck.
Fast Modular Transforms.
Journal of Computer and System Sciences 8, 366–386, 1974.

C. Fiduccia.
Polynomial evaluation via the division algorithm: The fast Fourier transform revisited.
In Proc. 4th Symp. on Theory of Computing, ACM Press, 88–93, 1972.

E. Horowitz.
A fast method for interpolation of polynomials using preconditioning.
Information Processing Letters 1, 157–163, 1972.

J. Lipson.
Chinese remainder and interpolation algorithms.
In Proc. of the 2nd Syrup. on Symbolic and Algebraic Manipulation, ACM Press, 372–391,1971.

R. Moenck and A. Borodin.
Fast modular transforms via division.
In Proc. of the 13th Annual Syrup. on Switching and Automata Theory, Oct. 1972.

E. Kaltofen and L. Yagati.
Improved Sparse Multivariate Polynomial Interpolation Algorithms.
International Symposium on Symbolic and Algebraic Computation, 467–474, 1988.