Faster computation of roots of polynomials over $\mathbb{F}_q$

Michael Monagan
Simon Fraser University

ICIAM minisymposium on
Polynomial and Computational Challenges in Computer Algebra

Vancouver, July 18th, 2011
A polynomial interpolation problem

Application [MJ 2010]: to interpolate a polynomial in 12 variables of degree 30 with $t$ non-zero terms modulo a 32 bit prime $p$ we need to compute the roots of $\Lambda(z) \in \mathbb{F}_p[z]$ of degree $t$ using [Rabin 1980] where $\Lambda(z)$ has $t$ roots in $\mathbb{F}_p$.

<table>
<thead>
<tr>
<th>$t$</th>
<th>1 core</th>
<th>4 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>roots</td>
</tr>
<tr>
<td>1019</td>
<td>7.94</td>
<td>0.65</td>
</tr>
<tr>
<td>2041</td>
<td>31.3</td>
<td>2.47</td>
</tr>
<tr>
<td>4074</td>
<td>122.3</td>
<td>9.24</td>
</tr>
<tr>
<td>8139</td>
<td>484.6</td>
<td>34.7</td>
</tr>
</tbody>
</table>

Cilk timings in CPU seconds on an Intel Corei7

Ahmdal’s law ($t = 8139$): speedup $\leq 3.21$ (4 cores) and $\leq 6.31$ (12 cores).

We parallelized the solve time and reduced the roots sequential time from 34.7s to 10.4s (classical) to 2.25s (FFT) then 1.5s (GCD):

Ahmdal’s law ($t = 8139$): speedup $\leq 3.96$ (4 cores) and $\leq 11.58$ (12 cores).
A polynomial interpolation problem

Application [MJ 2010]: to interpolate a polynomial in 12 variables of degree 30 with \( t \) non-zero terms modulo a 32 bit prime \( p \) we need to compute the roots of \( \Lambda(z) \in \mathbb{F}_p[z] \) of degree \( t \) using [Rabin 1980] where \( \Lambda(z) \) has \( t \) roots in \( \mathbb{F}_p \).

<table>
<thead>
<tr>
<th>( t )</th>
<th>1 core</th>
<th>4 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>roots</td>
</tr>
<tr>
<td>1019</td>
<td>7.94</td>
<td>0.65</td>
</tr>
<tr>
<td>2041</td>
<td>31.3</td>
<td>2.47</td>
</tr>
<tr>
<td>4074</td>
<td>122.3</td>
<td>9.24</td>
</tr>
<tr>
<td>8139</td>
<td>484.6</td>
<td>34.7</td>
</tr>
</tbody>
</table>

Cilk timings in CPU seconds on an Intel Corei7

Ahmdal’s law \((t = 8139)\): speedup \(\leq 3.21\) (4 cores) and \(\leq 6.31\) (12 cores).

We parallelized the solve time and reduced the roots sequential time from 34.7s to 10.4s (classical) to 2.25s (FFT) then 1.5s (GCD):

Ahmdal’s law \((t = 8139)\): speedup \(\leq 3.96\) (4 cores) and \(\leq 11.58\) (12 cores).
Let \( a, b \in F[x] \).
Compute the quotient \( q \) and remainder \( r \) of \( a \div b \) such that

\[ a = bq + r. \]

Let \( b = b_0 + b_1x + \cdots + b_dx^d \) so \( d = \deg b \).
Let \( b_r = b_d + \cdots + b_1x + b_0x^d \) be the reciprocal polynomial.

1: compute \( b_r^{-1} \) to \( O(x^{dq+1}) \) with a Newton iteration \( \ldots 2M(d) \)
2: compute \( q_r = \lfloor a_r b_r^{-1} \rfloor_{dq} \) then \( \ldots \ldots \ldots \ldots 1M(d) \)
3: compute \( r = a - bq \) \( \ldots \ldots \ldots \ldots \ldots \ldots 1M(d) \)
Inverse using a Newton Iteration

Input: \( d \in \mathbb{N} \) and \( b = b_0 + b_1x + \cdots \in F[x] \).
Compute \( y = b^{-1} \) to \( O(x^d) \)

1. \textbf{if} \( d = 1 \) \textbf{return} \( b_0^{-1} \).
2. compute \( y = b^{-1} \) to \( O(x^{\lceil d/2 \rceil}) \) recursively.
3. \textbf{return} \( (2y - y^2b) \mod x^d \).

MCA: \( T(d) = T\left(\frac{d}{2}\right) + M\left(\frac{d}{2}\right) + M(d) + O(d) \implies T(d) < 3M(d) \)
FFT: \( T(d) = T\left(\frac{d}{2}\right) + 3\text{FFT}(2d) + O(d) \implies T(d) < 2M(d) \)

\( \equiv 1M(d) \)
Inverse using a Middle Product

3 return $2y - yb^2 \mod x^d$.
3 return $y + y(1 - yb) \mod x^d$.

$yb = 1 + 0x + \cdots + 0x^{d-1} + \square x^{\frac{d}{2}} + \cdots + \square x^{d-1} + \square x^d + \cdots + \square x^{\frac{3}{2}d-2}$

HQZ [2002]:

$T(d) = T\left(\frac{d}{2}\right) + M\left(\frac{d}{2}\right) + MP\left(\frac{d}{2}\right) + O(d) \equiv 1M\left(\frac{d}{2}\right)$

$T(d) < 2M(d)$

FFT:

$T(d) = T\left(\frac{d}{2}\right) + 3FFT\left(\frac{3}{2}d\right) + O(d) \equiv M\left(\frac{3}{2}d\right)$

$T(d) < \frac{3}{2}M(d)$
Rabin’s 1980 root finding algorithm over $\mathbb{F}_q$

Input: $p$ an odd prime, $f = 1x^d + \cdots + f_1x + f_0 \in \mathbb{F}_p[x]$, $f_0 \neq 0$

Output: the roots of $f(x)$ in $\mathbb{F}_p$.

Lemma (Fermat)

Over $\mathbb{F}_p$, $x^{p-1} - 1 = \prod_{i=1}^{p-1}(x - i) = (x^{(p-1)/2} - 1)(x^{(p-1)/2} + 1)$

1. Compute $b = \gcd(x^{p-1} - 1, f) = \text{all linear factors of } f$.
2. If $\deg b > 1$ compute $h = \gcd((x + \alpha)^{(p-1)/2} - 1, b)$
   for random $\alpha \in \mathbb{F}_p$ until $h$ splits $b$.
   Then compute the roots of $h$ and $b/h$ recursively.

How do we compute $h = \gcd(a^m + c, b)$?
First compute $a^m \mod b$ using square-and-multiply.
Rabin’s 1980 root finding algorithm over $\mathbb{F}_q$

Input: $p$ an odd prime, $f = 1x^d + \cdots + f_1x + f_0 \in \mathbb{F}_p[x]$, $f_0 \neq 0$  
Output: the roots of $f(x)$ in $\mathbb{F}_p$.

Lemma (Fermat)

Over $\mathbb{F}_p$, $x^{p-1} - 1 = \prod_{i=1}^{p-1} (x - i) = (x^{(p-1)/2} - 1)(x^{(p-1)/2} + 1)$

1. Compute $b = \gcd(x^{p-1} - 1, f) =$ all linear factors of $f$.
2. If $\deg b > 1$ compute $h = \gcd((x + \alpha)^{(p-1)/2} - 1, b)$ for random $\alpha \in \mathbb{F}_p$ until $h$ splits $b$.
   Then compute the roots of $h$ and $b/h$ recursively.

How do we compute $h = \gcd(a^m + c, b)$ ?
First compute $a^m \mod b$ using square-and-multiply.
Algorithm Square-and-Multiply modulo $b(x) \in F[x]$

Input: $m \in \mathbb{N}$ and $a, b \in F[x]$ of degree $\deg a < d = \deg b$.
Output: $r = a^m \mod b$.

set $r = a$ and let $m = m_l \cdots m_2m_1$ in binary.

for $k = l - 1$ downto 1 do

set $s = r^2$ ........................................... $1M(d)$
set $r = s \mod b$ ........................................... $4M(d)$

if $m_k = 1$ set $r = ar \mod b$ ...... $(a = x + \alpha)$ ...... $O(d)$

return $r$

Costs $5M(d)$ per iteration.

MCA: $3M(d)$ by precomputing $b_r^{-1}$.
MCA: $2M(d)$ by precomputing $\text{FFT}_\omega(b_r^{-1})$ and $\text{FFT}_\omega(b_r)$.
MBM: $1M(d)$ by staying in FFT co-ordinates.
Algorithm Square-and-Multiply modulo $b(x) \in F[x]$

Input: $m \in \mathbb{N}$ and $a, b \in F[x]$ of degree $\deg a < d = \deg b$.
Output: $r = a^m \mod b$.

set $r = a$ and let $m = m_l \cdots m_2 m_1$ in binary.
for $k = l - 1$ downto 1 do
    set $s = r^2$ ....................................................... $1M(d)$
    set $r = s \mod b$ ................................................... $4M(d)$
    if $m_k = 1$ set $r = ar \mod b$ ...... ($a = x + \alpha$) ...... $O(d)$
return $r$

Costs $5M(d)$ per iteration.
MCA: $3M(d)$ by precomputing $b_r^{-1}$.
MCA: $2M(d)$ by precomputing $FFT_\omega(b_r^{-1})$ and $FFT_\omega(b_r)$.
MBM: $1M(d)$ by staying in FFT co-ordinates.
First idea: precompute $\text{FFT}_\omega(b_r^{-1})$ and $\text{FFT}_\omega(b_r)$

\begin{align*}
\text{set } s &= r^2 \quad \text{.......................... } 2 \text{ FFTs} \\
\text{set } d_q &= 2d_r - d. \\
\text{if } d_q \geq 0 \text{ then compute } r = s \mod b: \\
&\quad \text{set } t = \lfloor s \rfloor_{d_q} \quad \text{.......................... } O(d) \\
&\quad \text{set } q_r = t_r \cdot b_r^{-1} \quad \text{.......................... } 2 \text{ FFTs} \\
&\quad \text{set } q_r = \lfloor q_r \rfloor_{d_q} \quad \text{.......................... } O(d) \\
&\quad \text{set } r_r = s_r - b_r q_r \quad \text{.......................... } 2 \text{ FFTs} \\
&\quad \quad r_r = [0, 0, \ldots, 0, □, □, \ldots, □] \\
&\quad \quad \quad \quad dq+1 \text{ zeroes remainder} \\
&\quad \text{set } r_r = r_r/x^{dq+1} \quad \text{.......................... } O(d) \\
&\quad \text{set } d_r = \text{deg } r \quad \text{.......................... } O(d)
\end{align*}

We have 6 FFTs of degree $< 2d \equiv 2M(d)$. 

Michael Monagan

Faster computation of roots of polynomials over $F_q$
Main idea: stay in FFT co-ordinates

\[
\text{set } s_r = r_r^2 \\
\text{set } d_q = 2d_r - d. \\
\textbf{if } d_q \geq 0 \textbf{ then compute } r = s \mod b: \\
\text{set } t_r = \lfloor s_r \rfloor_{d_q} \\
\text{set } q_r = t_r \cdot b_r^{-1} \\
\text{set } q_r = \lfloor q_r \rfloor_{d_q} \\
\text{set } r_r = s_r - b_r q_r \\
\text{set } d_r = \deg r \text{ ???}
\]

We have 4 FFTs of degree \( < 2d \equiv \frac{4}{3} M(d) \).
Main idea: stay in FFT co-ordinates

\[ s_r = r_r^2 \]  
\[ d_q = 2d_r - d. \]

If \( d_q \geq 0 \)
compute \( r = s \mod b \):

set \( t_r = \lfloor s_r \rfloor_{d_q} \)
set \( r_r = s_r - b_r q_r \)

\[ r_r = [0, \square, \ldots, \square], \square, \ldots, \square] \]
remainder of degree \( d-2 \)

set \( d_r = d - 1 \)

\[ d_q = 2d_r - d. \]
Final idea: do 2 larger FFTs

\[ s_r = r_r^2 \] \hspace{2cm} O(d)

\[ d_q = 2d_r - d. \]

**if** \( d_q \geq 0 \) **then** compute \( r = s \mod b: \)

// set \( t_r = \lfloor s_r \rfloor_{d_q} \) .... \( s_r = [0,0,\square,\ldots,\square] \) ..... OMIT

\[ q_r = s_r \cdot b_r^{-1} \] .... \( q_r = [0,0,\square,\ldots,\square] \) ..... \( O(d) \)

\[ q_r = \lfloor q_r \rfloor_{d_q} \text{ ( has degree } < 3d \text{ )} \] ............ \( 2 \) FFTs

**if** \( \delta > 0 \) **set** \( d_q = d_q - \delta \) and \( s_r = s_r / x^\delta \)

\[ r_r = s_r - b_rq_r \] \hspace{2cm} O(d)

\[ r_r = r_r / x^{dq+1} \] \hspace{2cm} O(d)

\[ d_r = d - 1 \]

We have 2 FFTs of degree < 3d \( \equiv 1M(d). \)
A benchmark

Compute the $d - 3$ roots of $f(x) = (x^d - 1)/(x^2 - 1)$ in $\mathbb{F}_p$ for $d = 2^k$ where $p = 2^{201017} + 1$.

<table>
<thead>
<tr>
<th>$d$</th>
<th>Maple 14</th>
<th>Mahdi</th>
<th>Magma</th>
<th>Classical</th>
<th>FFT</th>
<th>Lehmer</th>
<th>GCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>24.0s</td>
<td>9.2s</td>
<td>7.0s</td>
<td>2.55s</td>
<td>0.8s</td>
<td>0.58s</td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>96.3s</td>
<td>34.7s</td>
<td>17.2s</td>
<td>10.4s</td>
<td>2.3s</td>
<td>1.50s</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>339.7s</td>
<td>48.9s</td>
<td></td>
<td>39.4s</td>
<td>7.2s</td>
<td>4.2s</td>
<td></td>
</tr>
</tbody>
</table>

Maple is using classical polynomial arithmetic $O(\log(p)d^2 \log p)$. Magma is using fast polynomial arithmetic $O(d \log^2 d \log p)$.
Current and Future Work

- fast Euclidean algorithm for GCD [Soo Go]
- parallelize the 4 multiplications inside the fast Euclidean algorithm
- need parallel FFT for large $d$
- after splitting $f(x)$ compute the roots recursively in parallel
### Maple 14 code

<table>
<thead>
<tr>
<th>Maple 14 code</th>
<th>Magma code</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; p := 2114977793;</td>
<td>&gt; p := 2114977793;</td>
</tr>
<tr>
<td>&gt; Fp := GaloisField(p);</td>
<td>&gt; Fp := GaloisField(p);</td>
</tr>
<tr>
<td>&gt; Zpx&lt;x&gt; := PolynomialRing(Fp);</td>
<td>&gt; Zpx&lt;x&gt; := PolynomialRing(Fp);</td>
</tr>
<tr>
<td>&gt; d := 8192;</td>
<td>&gt; d := 8192;</td>
</tr>
<tr>
<td>&gt; divide(x^d-1,x^2-1,’f’);</td>
<td>&gt; f := ExactQuotient(x^d-1,x^2-1);</td>
</tr>
<tr>
<td>&gt; nops(Roots(f) mod p);</td>
<td>&gt; #Roots(f);</td>
</tr>
<tr>
<td>&gt; quit;</td>
<td>&gt; quit;</td>
</tr>
</tbody>
</table>