Processor-efficient exponentiation in finite fields *

Joachim Von Zur Gathen
Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S IA4

Communicated by T. Lengauer
Received 24 June 1991
Revised 23 October 1991

Abstract
The processor-efficiency of parallel algorithms for exponentiation in a finite field extension is studied, assuming that a normal basis over the ground field is given.

Keywords: Design of algorithms, parallel algorithms, finite field arithmetic, cryptography

1. Introduction

In this paper, we study the number of processors used for parallel exponentiation in finite fields. This problem is important in some cryptographic applications.

Let us first describe the problem. We consider exponentiation in a finite field \( F_q^n \) with \( q^n \) elements, where \( q \) is a prime power and \( n \geq 1 \). Suppose that \( (\beta_0, \ldots, \beta_{n-1}) \) is a normal basis of \( F_q^n \) over a field \( F_q \) with \( q \) elements, so that \( \beta_i^q = \beta_i \) for all \( i \). An arbitrary element \( a \) of \( F_q^n \) can be uniquely written as \( a = \sum_{0 \leq i < n} a_i \beta_i \), with \( a_0, \ldots, a_{n-1} \in F_q \). For any \( j \in \mathbb{N} \), we have

\[
\sum_{0 \leq i < n} a_i \beta_i^j = \sum_{0 \leq i < n} a_i \beta_{i+j}\,
\]

with index arithmetic modulo \( n \). Thus taking \( q^{th} \) powers amounts to a cyclic shift of coordinates.

This may be much less expensive than a general multiplication.

A basic assumption for our algorithms is that computing \( q^{th} \) powers is for free. This assumption is used in the literature for \( q = 2 \) [1,3,6,9]. Normal bases are easy to find (see [8] and the literature given there).

Section 2 presents the basic algorithm that we will use. It works in size (= total number of multiplications) and width (= number of processors) about \( n/\log_q n \) and depth (= parallel time = delay) about \( \log_2 n \). Section 3 analyses the width of the algorithm, shows how it can sometimes be reduced by appropriate load-balancing, and gives some numerical values. In Section 4, the algorithm is discussed under the assumption that only few processors are available.

2. Multiplication and free powers

The algorithms we will consider only use multiplication and \( q^{th} \) powers; the latter are assumed to have zero cost. It is convenient to think of such an algorithm as a directed acyclic graph,
or arithmetic circuit. Three measures are of interest: The depth (= parallel time) is the maximal length (= number of multiplication gates) of paths in such a circuit, and the size (= total work) is the number of multiplication gates. We can stratify the circuit into levels, with input and constant gates at level zero, and any other gate at a higher level than any of its two inputs. Then the width (= number of processors) of a circuit is the maximal number of gates at any level. Trivially, we have

\[ \text{width} \leq \text{size} \leq \text{depth} \cdot \text{width}. \quad (2.1) \]

We want to compute \( x^e \in \mathbb{F}_q[x] \). Since \( a^{q^n} = a \) for all \( a \in \mathbb{F}_q \), by Fermat’s Little Theorem, we may assume that our exponent \( e \) satisfies \( 0 \leq e < q^n \). We take the \( q \)-ary representation of \( e \)

\[ e = \sum_{0 \leq i < n} e_i q^i \]

with \( 0 \leq e_0, \ldots, e_{n-1} < q \).

Then \( x^e \) can obviously be computed as follows.

**Algorithm 1**

1. For \( 2 \leq j < q \), compute \( x^j \).
2. For \( 1 \leq i < n \), compute \( y_i = (x^{e_i})^{q^j} \).
3. Compute the product \( x^e = \prod_{0 \leq i < n} y_i \).

This algorithm uses depth \( \delta + [\log_2 n] \), width \( \max(2^\delta - 2, q - 1 - 2^{\delta - 1}, [n/2]) \), and size \( q + n - 3 \), where \( \delta = [\log_2 (q - 1)] \).

The idea of the next algorithm is that short patterns might occur repeatedly in the \( q \)-ary representation of \( e \), and that precomputation of all such patterns might lower the overall cost. This idea is useful for “addition chains” (see [4, 6, 3], and the references given there) and for “word chains” [2], and has been applied to our exponentiation problem in characteristic two, by Agnew et al. [1] and Stinson [6].

We choose some pattern length \( r \geq 1 \), set \( s = [n/r] \), and write \( e = \sum_{0 \leq i < s} b_i q^r^i \) with \( 0 \leq b_i < q^r \) for all \( i \).

**Algorithm 2**

1. For \( 2 \leq d < q \), compute \( x^d \).
2. For \( q < d < q^r \), compute \( x^d \).
3. For \( 0 \leq i < s \), compute \( y_i = (x^{b_i})^{q^r^i} \).
4. Return \( x^e = \prod_{0 \leq i < s} y_i \).

Step 1 takes depth \( \delta + [\log_2(q - 1)] \). We implement step 2 in \( [\log_2 r] \) stages 1,...,\( [\log_2 r] \) as follows. For any \( d \in \mathbb{N} \) with \( q \)-ary representation \( d = \sum d_i q^i \), let

\[ w(d) = \#\{i: d_i \neq 0\} \quad (2.2) \]

be the \( q \)-ary Hamming weight of \( d \). In stage \( i \), we compute all \( x^d \) (not previously computed) with \( q < d < q^r \), \( d \neq 0 \mod q \), and \( w(d) < 2^i \). Each new \( d \) in stage \( i \) is of the form \( d = d_1 + q^j d_2 \), for some \( j \geq 1 \) and \( d_1, d_2 \) computed before stage \( i \).

Then

\[ x^d = x^{d_1} \cdot (x^{d_2})^{q^j}, \]

and each stage \( 1, \ldots, [\log_2 r] \) can be performed in depth 1.

After the last stage, we have all required powers for \( d \neq 0 \mod q \); the ones with \( d = 0 \mod q \) can be computed free of charge. There are exactly \( q^r - q^{r-1} - 1 \) integers \( d \) with \( 2 \leq d < q^r \) and \( d \neq 0 \mod q \). Since each multiplication yields a new \( x^d \), the total size for steps 1 and 2 is \( (q - 1)q^{r-1} - 1 \). This also bounds the width.

Step 3 is free, and we can use a binary multiplication tree in step 4. Together we obtain depth

\[ [\log_2(q - 1)] + [\log_2 r] + [\log_2 n], \]

width \( \max((q - 1)q^{r-1} - 1, [s/2]) \), and size \( (q - 1)q^{r-1} + s - 2 \).

In the sequel, we study the width in more detail. For sufficiently large \( n \), the choice

\[ r = [\log_2 n - 2 \log_2 \log_2 n] \]

leads to width less than

\[ \frac{n}{2 \log_2 n} \left( 1 + (10 \log_2 \log_2 n + 6)/\log_2 n \right) \]

and size at most twice that bound. This size is optimal up to a factor of three [4], and no smaller size is known; these facts will not be used.
3. Balancing the last two stages

We want to examine the width of Algorithm 2 in more detail. For $a, b, q, r \in \mathbb{N}$, define

$$
\Delta_{q,r}(a, b) = \sum_{a \leq k < b} \binom{r-1}{k} (q-1)^{k+1}.
$$

Then we have

$$
\Delta_{q,r}(a, b) = \sum_{a \leq c < b} \Delta_{q,r}(c, c+1),
$$

and

$$
\Delta_{q,r}(0, r) = (q-1)^{r-1}.
$$

There are $\Delta_{q,r}(a, b)$ many $d$ with $1 \leq d < q'$, $d \neq 0 \mod q$, $a < w(d) < b$. This is easily verified for $b = a + 1$, and follows in general from (3.1).

For $1 \leq \mu < \lambda = \lfloor \log_2 r \rfloor$, the width at stage $\mu$ of step 2 is $\Delta_{q,r}(2^{\mu-1}, 2^\mu)$. We first show that the maximum occurs for $\mu = \lambda - 1$. We have $2^{\lambda-2} \leq (r-1)/2 < 2^\lambda$, and thus the middle term $m = (r-1)/2$ of the binomial expansion (3.1) contributes to $\Delta_{q,r}(2^{\lambda-2}, 2^\lambda)$ unless $r = 2^\lambda$. Thus for $k$ with $2^{\lambda-3} \leq k < 2^{\lambda-2}$ can be matched bijectively with an interval in $[2^{\lambda-2}, 2^{\lambda-1}]$ starting or ending with $m$, and therefore

$$
\Delta_{q,r}(2^{\lambda-3}, 2^{\lambda-2}) \leq \Delta_{q,r}(2^{\lambda-2}, 2^{\lambda-1}).
$$

(This also holds when $r = 2^\lambda$.) More generally, we have

$$
\Delta_{q,r}(2^{\mu-1}, 2^{\mu}) \leq \Delta_{q,r}(2^{\lambda-2}, 2^{\lambda-1})
$$

for $1 \leq \mu < \lambda$. Thus the maximal width in step 2 of Algorithm 2 occurs at stage $\lambda - 1$ or $\lambda$, and the total width in the algorithm is in fact equal to

$$
\max\{\Delta_{q,r}(2^{\lambda-2}, 2^{\lambda-1}), \Delta_{q,r}(2^{\lambda-1}, r), \lfloor s/2 \rfloor\}.
$$

### Table 1

$q = 3, n = 1000$

<table>
<thead>
<tr>
<th>depth</th>
<th>width</th>
<th>size</th>
<th>$r$</th>
<th>$s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>11</td>
<td>500</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>11</td>
<td>167</td>
<td>351</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>125</td>
<td>302</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>143</td>
<td>360</td>
<td>5</td>
</tr>
</tbody>
</table>

### Table 2

$q = 2, n = 593$

<table>
<thead>
<tr>
<th>depth</th>
<th>width</th>
<th>size</th>
<th>$r$</th>
<th>$s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stinson [6]</td>
<td>10</td>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm 1</td>
<td>10</td>
<td>296</td>
<td>592</td>
<td></td>
</tr>
<tr>
<td>Algorithm 2 with (3.2)</td>
<td>10</td>
<td>49</td>
<td>129</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>42</td>
<td>147</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>64</td>
<td>201</td>
<td>8</td>
</tr>
</tbody>
</table>

### Table 3

$q = 2, n = 1013$

<table>
<thead>
<tr>
<th>depth</th>
<th>width</th>
<th>size</th>
<th>$r$</th>
<th>$s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>10</td>
<td>84</td>
<td>199</td>
<td>6</td>
</tr>
<tr>
<td>Algorithm 2 with (3.2)</td>
<td>11</td>
<td>73</td>
<td>207</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>64</td>
<td>253</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>162</td>
<td>367</td>
<td>9</td>
</tr>
</tbody>
</table>

In general, we do the following. Let

$$
D = \{d: 2 \leq d < 2^r, d = 1 \mod 2\}
$$

be the set of all exponents $d$ for which $x^d$ has to be computed in step 2. Thus $\#D = 2^{r-1} - 1$.

We assume $\lambda \geq 3$. Up to and including stage $\lambda - 2$, $t_2 = \Delta_{q,r}(1, 2^{\lambda-2})$ of the $d \in D$ have been dealt with. Since we cannot more than double the
weight in one step, at most \( m_2 = \Delta_{2r_2}(2^{\lambda - 2}, 2^{\lambda - 1}) \)
d's can be finished at stage \( \lambda - 1 \). This is done in
Algorithm 2 of Section 2, and is the best we can
do if \( r = 2^k \). However, in some cases we can
balance the work better by dealing only with the
\[
t_1 = \min\{m_2, 2^{r - 2} - \lfloor \frac{t_2}{2} \rfloor - 1\}
\]
many new \( d \in D \) of smallest weight at stage \( \lambda - 1 \).
Then at stage \( \lambda \) only
\[
t_0 = 2^{r - 1} - 1 - t_1 - t_2
\]
\[
= \max\{2^{r - 1} - 1 - m_2 - t_2, 2^{r - 2} - \lfloor \frac{t_2}{2} \rfloor\}
\]
operations have to be performed.

**Theorem 3.1.** Let \( q = 2 \), \( 0 \leq e < 2^r \), \( r \geq 5 \), \( s = \lfloor n/r \rfloor \), and \( t_0 \) as above. The algorithm given above computes \( x^e \) in depth
\( \lceil \log_2 r \rceil + \lceil \log_2 s \rceil \), width
\( \max\{t_0, \lfloor s/2 \rfloor\} \), and size \( 2^{r - 1} + s - 2 \).

**Proof.** Stage \( \lambda - 1 \) can be performed in depth 1
since \( t_1 \leq m_2 \). To prove the same for stage \( \lambda \),
we have to check that every \( d \in D \) with \( w(d) \leq \lfloor r/2 \rfloor \)
is dealt with up to stage \( \lambda - 1 \). Let \( m_1 = \Delta_{2,r_2}(2^{\lambda - 2}, \lfloor r/2 \rfloor) \). It is sufficient to show \( m_1 \leq t_1 \).
This is clear if \( t_1 = m_2 \), since \( r/2 \leq 2^{\lambda - 1} \). If \( t_1 \neq m_2 \),
we find from the binomial expansion of \( 2^{r - 1} \)
\[
2t_2 + 2m_1 = 2 \cdot \Delta_{2,r_2}(1, \lfloor r/2 \rfloor) < 2^{r - 1},
\]
\[
m_1 < 2^{r - 2} - t_2, < t_1.
\]
The depth claimed in the theorem now follows.

Stage \( \lambda - 2 \) uses width \( m_3 = \Delta_{2,r_2}(2^{\lambda - 2}, 2^{\lambda - 2}) < m_2/2 \), and each previous stage uses smaller
width. Since
\[
t_1 \leq 2^{r - 2} - \lfloor \frac{t_2}{2} \rfloor - 1 < 2^{r - 2} - \lfloor \frac{t_2}{2} \rfloor \leq t_0,
\]
\[
m_2 + 1 \leq m_2 + \Delta_{2,r_2}(2^{\lambda - 1}, r)\]
\[
= m_2 + \left(2^{r - 1} - 1 - t_2 - m_2\right)\]
\[
= t_0 + t_1 \leq 2t_0,
\]
the claimed bound on the width follows. \( \Box \)

As a further example, consider the second last line (\( r = 9 \)) of Table 3. Stage 3 performs 162
multiplications, and stage 4 only 1. The balancing
reduces the width to 82.

4. **Using few processors**

There exist standard rescheduling techniques
for using the type of algorithm discussed here
with fewer processors than the stated width, by
distributing the computations of one “level” onto
several levels, thus increasing the depth and
reducing the width (see, e.g., [6]).

If we have \( w \) processors available, we can
calculate the product of \( s \) factors as follows. In
each of \( d = \lfloor s/w \rfloor - 1 \) stages, \( w \) pairs of factors
(from the given \( s \) ones or from previous stages)
are multiplied together. (We use \( d = 0 \) if \( s < w \).
This leaves \( s - dw < 2w \) factors, which are then
multiplied along a binary tree, of depth \( \lceil \log_2 (s - dw) \rceil \).
The total depth is \( d + \lceil \log_2 (s - dw) \rceil \). For our
standard example \( q = 2, n = 593 \), this yields

<table>
<thead>
<tr>
<th>depth</th>
<th>width</th>
<th>size</th>
<th>( r )</th>
<th>( s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>2</td>
<td>129</td>
<td>6</td>
<td>99</td>
</tr>
<tr>
<td>75</td>
<td>2</td>
<td>147</td>
<td>7</td>
<td>85</td>
</tr>
<tr>
<td>34</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>8</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>16</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>32</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>32</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^{a}\) The size and \( s \) depend only on \( r \).
the depths in Table 5 for widths that are powers of two. Again, this compares favorably with Stinson's [6] estimates of depth 77, 29, 15, and 11 for widths 4, 8, 16, and 32, respectively.

In fact, this depth is optimal. To see this, consider a computation of \(x_1 \cdots x_n\) in width \(w\) and depth \(\delta\). We may assume \(s > 2w\), since otherwise \(d = 0\) and the binary tree of depth \([\log_2 s]\) is optimal. Since the fan-in is two and there is a single output node, one sees by induction on \(i\) that there are at most \(2^i\) multiplications at depth \(\delta - i\), for \(0 \leq i < m = [\log_2 w]\). Thus on the last \(m\) levels a total of at most \(\sum_{0 \leq i < m} 2^i = 2^m - 1\) multiplications is performed. At most \(w\) can be done on any single level, and a total of at least \(s - 1\) is required. Thus

\[
\delta \geq m + \left\lceil \frac{s - 1 - (2^m - 1)}{w} \right\rceil.
\]

Set \(u = [(s - 2^m)/w]\) and \(\ell = [\log_2 (s - dw)]\), with \(d = [s/w] - 1\). It is sufficient to show that

\[
m + u \geq \ell + d.
\]

Since \(w < s - dw < 2w\), we have \(m \leq \ell \leq m + 1\). We distinguish three cases.

If \(w\) divides \(s\), then \(d = s/w - 1\) and \(\ell = m\). Since \(-2^m/w > -2\), we have

\[
m + u = m + \frac{s}{w} + \left\lceil \frac{-2^m}{w} \right\rceil \geq m + \frac{s}{w} - 1 = \ell + d.
\]

If \(w\) does not divide \(s\) and \(\ell = m\), then

\[
u \geq \left\lceil \frac{s - 2w}{w} \right\rceil = \left\lceil \frac{s}{w} \right\rceil - 2 = \left\lfloor \frac{s}{w} \right\rfloor - 1 = d.
\]

If \(\ell = m + 1\), then

\[
\frac{s - 2^m}{w} = \frac{s - 2^{\ell - 1}}{w} > \frac{s - 2^{\log_2(s - dw)}}{w} = d,
\]

\[
u \geq d + 1.
\]

This proves that the depth is indeed optimal.

It is sometimes the case that faster algorithms have a higher overhead. This is not the case in steps 3 and 4 of Algorithm 2, which have the same regular structure as the simple Algorithm 1. (We remark that, skipping step 3, the multiplication tree in step 4 may be arranged so that at level \(i\) only \(q^2\)th powers are used, for \(1 \leq i \leq \log_5 s\).) Steps 1 and 2 are somewhat more complicated, but independent of the exponent \(e\). If arithmetic in one fixed field extension is required, these steps can be pre-programmed, in software or maybe even in hardware.

Following a proposal by Agnew et al. [1] (for \(q = 2\)) we might arrange steps 3 and 4 of Algorithm 2 according to

\[
e = \sum_{0 \leq i < s} b_i q^{ri} = \sum_{0 \leq i < q^r} c \left( \sum_{i \equiv i_c} q^i \right),
\]

where \(I_c = \{i: 0 \leq i < s, b_i = c\}\). Let \(m_c = \# I_c\). This version leads to depth

\[
d_c = \max_{0 \leq i < q^r} \left[ \log_2 m_c \right] + \left[ \log_2 q^r \right],
\]

width

\[
w = \max \left\{ \left[ q^r / 2 \right], \sum_{0 \leq i < q^r} \left[ m_c / 2 \right] \right\},
\]

and size \(s - 1\). We have

\[
\log_2 s \leq d \leq \log_2 s + r \log_2 q,
\]

\[
\frac{s - q^r}{2} \leq \frac{w}{2} \leq \frac{s}{2}.
\]

If each \(m_c\) is about \(s/q^r\), this yields slightly smaller width without increasing the depth. In general, we cannot expect the \(m_c\)'s to be of equal size, and then the increase in depth can be more advantageously used to reduce the width by the balancing technique as in Table 5.

References


