Information Processing Letters 41 (1992) 81–86
North-Holland

14 February 1992

Processor-efficient exponentiation in finite fields *

Joachim Von Zur Gathen

Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4

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

Abstract

Von Zur Gathen, J., Processor-efficient exponentiation in finite fields, Information Processing Letters 41 (1992) 81–86.

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 $\mathbb{F}_{q^n}$ with q^n elements, where q is a prime power and $n \geqslant 1$. Suppose that $(\beta_0, \ldots, \beta_{n-1})$ is a *normal basis* of F over a field $\mathbb{F}_q$ with q elements, so that $\beta_i = \beta_0^{q^i}$ for all i. An arbitrary element a of $\mathbb{F}_{q^n}$ can be uniquely written as $a = \sum_{0 \leqslant i < n} a_i \beta_i \in F$ with $a_0, \ldots, a_{n-1} \in \mathbb{F}_q$. For any $j \in \mathbb{N}$, we have

$$a^{q^j} = \sum_{0 \leqslant i < n} a_i \beta_i^{q^j} = \sum_{0 \leqslant i < n} a_i \beta_{i+j},$$

with index arithmetic modulo n. Thus taking qth 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 qth 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 qth powers; the latter are assumed to have zero cost. It is convenient to think of such an algorithm as a directed acyclic graph,

* Part of this work was done while the author was a Visiting Fellow at the Computer Sciences Laboratory, Australian National University, Canberra, Australia, and partly supported by Natural Sciences and Engineering Research Council of Canada, grant A2514.

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

$$\text{width} \leqslant \text{size} \leqslant \text{depth} \cdot \text{width}. \tag{2.1}$$

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

$$e = \sum_{0 \leqslant i < n} e_i q^i \quad \text{with } 0 \leqslant e_0, \ldots, e_{n-1} < q.$$

Then x^e can obviously be computed as follows.

Algorithm 1

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

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

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, 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 $r \geqslant 1$, set $s = \lceil n/r \rceil$, and write $e = \sum_{0 \leqslant i < s} b_i q^{ri}$ with $0 \leqslant b_i < q^r$ for all i.

Algorithm 2

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

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

$$w(d) = \#\{i \colon d_i \neq 0\} \tag{2.2}$$

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

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

and each stage $1, \ldots, \lceil \log_2 r \rceil$ can be performed in depth 1.

After the last stage, we have all required powers for $d \not\equiv 0 \mod q$; the ones with $d \equiv 0 \mod q$ can be computed free of charge. There are exactly $q^r - q^{r-1} - 1$ integers d with $2 \leqslant d < q^r$ and $d \not\equiv 0 \mod q$. 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

$$\lceil \log_2(q-1) \rceil + \lceil \log_2 r \rceil + \lceil \log_2 s \rceil,$$

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

$$r = \lfloor \log_q n - 2 \log_q \log_q n \rfloor$$

leads to width less than

$$\frac{n}{2 \log_q n}\left(1 + (10 \log_q \log_q n + 6)/\log_q n\right)$$

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

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

Then we have

$$\Delta_{q,r}(a, b) = \sum_{a \leqslant c < b} \Delta_{q,r}(c, c+1),$$
$$\Delta_{q,r}(0, r) = (q-1)q^{r-1}. \tag{3.1}$$

There are $\Delta_{q,r}(a, b)$ many d with $1 \leqslant d < q^r$, $d \not\equiv 0 \bmod q$, $a < w(d) \leqslant b$. This is easily verified for $b = a+1$, and follows in general from (3.1).

For $1 \leqslant \mu < \lambda = \lceil \log_2 r \rceil$, the width at stage μ 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 $2^{\lambda-2} \leqslant (r-1)/2 < 2^{\lambda-1}$, and thus the middle term $m = \lceil (r-1)/2 \rceil$ of the binomial expansion (3.1) contributes to $\Delta_{q,r}(2^{\lambda-2}, 2^{\lambda-1})$ unless $r = 2^{\lambda}$. Thus the k with $2^{\lambda-3} \leqslant k < 2^{\lambda-2}$ can be matched bijectively with an interval in $[2^{\lambda-2}, 2^{\lambda-1}-1]$ starting or ending with m, and therefore

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

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

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

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

$$\max\{\Delta_{q,r}(2^{\lambda-2}, 2^{\lambda-1}),\ \Delta_{q,r}(2^{\lambda-1}, r),\ \lfloor s/2 \rfloor\}. \tag{3.2}$$

Table 1
$q = 3$, $n = 1000$

	depth	width	size	r	s
Algorithm 1	11	500	1000		
Algorithm 2	11	167	350	3	334
	10	125	302	4	250
	11	143	360	5	200

Table 2
$q = 2$, $n = 593$

	depth	width	size	r	s
Stinson [6]	10	49			
Algorithm 1	10	296	592		
Algorithm 2 with (3.2)	10	49	129	6	99
	10	42	147	7	85
	10	64	201	8	75

Tables 1, 2, 3, and 4 present a few examples, with small q. The field considered in Table 2 is used in a commercial cryptosystem [5].

We describe how one can reduce the width in step 2 of Algorithm 2 in some cases, while maintaining the depth. We only consider $q = 2$, although the approach will work in general. If $r = 2^{\lambda}$, where $\lambda = \lceil \log_2 r \rceil$ is the number of stages, we propose no reduction. In an extreme case, however, r might equal $2^{\lambda-1}+1$, and only very little work would be done at the last stage of step 1. As an example, consider $q = 2$, $n = 2048$, $r = 10$ as in the second to last line of Table 4. The work at stages 1, 2, 3, 4 of step 2 of the algorithm is 9, 120, 372, 10 multiplications, respectively. We can reduce the width to $(372 + 10)/2 = 191$ by distributing the work of the last two stages evenly between them.

In general, we do the following. Let

$$D = \{d : 2 \leqslant d < 2^r,\ d \equiv 1 \bmod 2\}$$

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 $\lambda \geqslant 3$. Up to and including stage $\lambda - 2$, $t_2 = \Delta_{2,r}(1, 2^{\lambda-2})$ of the $d \in D$ have been dealt with. Since we cannot more than double the

Table 3
$q = 2$, $n = 1013$

	depth	width	size	r	s
Algorithm 1	10	500	1012		
Algorithm 2 with (3.2)	11	84	199	6	169
	11	73	207	7	145
	10	64	253	8	127
	11	162	367	9	113
Theorem 3.1	11	82	367	9	113

Table 4
$q = 2$, $n = 2048$

	depth	width	size	r	s
Stinson [6]	11	225			
Algorithm 1	11	1024	2047		
Algorithm 2 with (3.2)	12	146	355	7	293
	11	128	382	8	256
	11	162	482	9	228
	11	372	715	10	205
Theorem 3.1	11	114	482	9	228
	11	191	715	10	205

weight in one step, at most $m_2 = \Delta_{2,r}(2^{\lambda-2}, 2^{\lambda-1})$ 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 $r = 2^\lambda$. However, in some cases we can balance the work better by dealing only with the

$$t_1 = \min\{m_2, 2^{r-2} - \lfloor t_2/2 \rfloor - 1\}$$

many new $d \in D$ of smallest weight at stage $\lambda - 1$. Then at stage λ only

$$t_0 = 2^{r-1} - 1 - t_1 - t_2$$
$$= \max\{2^{r-1} - 1 - m_2 - t_2, 2^{r-2} - \lceil t_2/2 \rceil\}$$

operations have to be performed.

Theorem 3.1. *Let* $q = 2$, $0 \leqslant e < 2^n$, $r \geqslant 5$, $s = \lceil n/r \rceil$, *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 $t_1 \leqslant m_2$. To prove the same for stage λ, we have to check that every $d \in D$ with $w(d) \leqslant \lceil r/2 \rceil$ is dealt with up to stage $\lambda - 1$. Let $m_1 = \Delta_{2,r}(2^{\lambda-2}, \lceil r/2 \rceil)$. It is sufficient to show $m_1 \leqslant t_1$. This is clear if $t_1 = m_2$, since $r/2 \leqslant 2^{\lambda-1}$. If $t_1 \neq m_2$, we find from the binomial expansion of 2^{r-1}

$$2t_2 + 2m_1 = 2 \cdot \Delta_{2,r}(1, \lceil r/2 \rceil) < 2^{r-1},$$
$$m_1 < 2^{r-2} - t_2 \leqslant t_1.$$

The depth claimed in the theorem now follows.

Stage $\lambda - 2$ uses width $m_3 = \Delta_{2,r}(2^{\lambda-3}, 2^{\lambda-2}) < m_2/2$, and each previous stage uses smaller width. Since

$$t_1 \leqslant 2^{r-2} - \lfloor t_2/2 \rfloor - 1 \leqslant 2^{r-2} - \lceil t_2/2 \rceil \leqslant t_0,$$
$$m_2 + 1 \leqslant m_2 + \Delta_{2,r}(2^{\lambda-1}, r)$$
$$= m_2 + (2^{r-1} - 1 - t_2 - m_2)$$
$$= t_0 + t_1 \leqslant 2t_0,$$
$$m_3 < m_2/2 < t_0,$$

the claimed bound on the width follows. □

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 5
Using few processors

depth	width	size [a]	r	s [a]
66	2	129	6	99
75	2	147	7	85
34	4		6	
39	4		7	
20	8		6	
21	8		7	
13	16		6	
14	16		7	
11	32		6	
10	32		7	

[a] The size and s depend only on r.

the depths in Table 5 for widths which are a power of two. Again, this compares favorably with Stinson's [6] estimates of depth 77, 29, 15, and 11 for width 4, 8, 16, and 32, respectively.

In fact, this depth is optimal. To see this, consider a computation of $x_1 \cdots x_s$ in width w and depth δ. We may assume $s \geqslant 2w$, since otherwise $d = 0$ and the binary tree of depth $\lceil \log_2 s \rceil$ 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 $0 \leqslant i < m = \lceil \log_2 w \rceil$. Thus on the last m levels a total of at most $\Sigma_{0 \leqslant i < m} 2^i = 2^m - 1$ multiplications is performed. At most w can be done on any single level, and a total of at least $s - 1$ is required. Thus

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

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

$$m + u \geqslant \ell + d.$$

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

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

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

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

$$u \geqslant \left\lceil \frac{s - 2w}{w} \right\rceil = \left\lceil \frac{s}{w} \right\rceil - 2 = \left\lfloor \frac{s}{w} \right\rfloor - 1 = d.$$

If $\ell = m + 1$, then

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

$$u \geqslant d + 1.$$

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^i}th powers are used, for $1 \leqslant i \leqslant \log_2 s$.) 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

$$e = \sum_{0 \leqslant i < s} b_i q^{ri} = \sum_{0 \leqslant c < q^r} c \left(\sum_{i \in I_c} q^{ri} \right),$$

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

$$d = \max_{0 \leqslant c < q^r} \lceil \log_2 m_c \rceil + \lceil \log_2 q^r \rceil,$$

width

$$w = \max\left\{ \lfloor q^r/2 \rfloor, \sum_{0 \leqslant c < q^r} \lfloor m_c/2 \rfloor \right\},$$

and size $s - 1$. We have

$$\log_2 s \leqslant d \leqslant \log_2 s + r \log_2 q,$$

$$\frac{s}{2} - \frac{q^r}{2} \leqslant w \leqslant \frac{s}{2}.$$

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

[1] G.B. Agnew, R.C. Mullin and S.A. Vanstone, Fast exponentiation in GF(2^n), in: C.G. Günther, ed., *Advances in Cryptology — EUROCRYPT'88*, Lecture Notes in Computer Science **330** (Springer, Berlin, 1988) 251–255.

[2] J. Berstel and S. Brlek, On the length of word chains, *Inform. Process. Lett.* **26** (1987) 23–28.

[3] T. Beth, B.M. Cook and D. Gollmann, Architectures for exponentiation in GF(2^n), in: A.M. Odlyzko, ed., *Advances in Cryptology — CRYPTO'86*, Lecture Notes in Computer Science **263** (Springer, Berlin, 1986) 302–310.

[4] D.E. Knuth, *The Art of Computer Programming, Vol.2, Seminumerical Algorithms* (Addison-Wesley, Reading, MA, 2nd ed., 1981).

[5] Newbridge Microsystems, CA34C168 Data Encryption Processor, 1989.

[6] D.R. Stinson, Some observations on parallel algorithms for fast exponentiation in GF(2^n), *SIAM J. Comput.* **19** (1990) 711–717.

[7] J. Von Zur Gathen, Efficient exponentiation in finite fields, in: *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, San Juan, PR, 1991.

[8] J. Von Zur Gathen and M. Giesbrecht, Constructing normal bases in finite fields, *J. Symbolic Comput.* **10** (1990) 547–570.

[9] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura and I.S. Reed, VLSI architectures for computing multiplications and inverses in GF(2^m), *IEEE Trans. Comput.* **34** (1985) 709–717.