Comparison of Bit and Word Level Algorithms for Evaluating Unstructured Functions over Finite Rings

B. Sunar* and D. Cyganski

Department of Electrical & Computer Engineering,
Worcester Polytechnic Institute,
Worcester, Massachusetts 01609, USA
{sunar, cyganski}@wpi.edu

Abstract. We study the problem of implementing multivariate functions defined over finite rings or fields as parallel circuits. Such functions are essential for building cryptographic substitution boxes and hash functions. We present a modification to Horner's algorithm for evaluating arbitrary n-variate functions defined over finite rings and fields. Our modification is based on eliminating redundancies in the multivariate version of Horner's algorithm which occur when the evaluation takes place over a small finite mathematical structure and may be considered as a generalization of Shannon's lower bound and Muller's algorithm to word level circuits. If the domain is a finite field $GF(p)$ the complexity of multivariate Horner polynomial evaluation is improved from $O(p^n)$ to $O(\frac{p^n}{2n})$. We prove the optimality of the presented algorithm. Our comparison of the bit level approach to the optimized word level approach yields an interesting result. The bit level algorithm is more efficient in both area consumption and time delay. This suggests that unstructured functions over finite rings or fields should be implemented using the bit-level approach and not the commonly used word level implementation style.

Keywords: Horner's method, polynomial evaluation, multivariate polynomials, word level, finite fields.

1 Introduction

Essentially all secret and public key schemes are based on arbitrary looking and highly nonlinear logic function families which are indexed by a fixed length key. In fact, it is expected that a proper cryptographic function is practically indistinguishable from a function that is randomly picked from the set of all functions of certain degree. This arbitrariness requires cryptographers to propose highly complex and structureless functions. A few examples are round subfunctions of

* Berk Sunar is funded by National Science Foundation research grants CAREER award ITR-ANI-0133297 and ITR-ANI-0112889.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 237–249, 2005.

hash algorithms (e.g. SHA-1 [6], MD5 [11] etc.), and substitution boxes in block and stream ciphers (e.g. DES [7], AES [8] etc.), or public-key schemes defined over polynomial rings or finite fields. Earlier constructions were given in terms of lookup tables which were built by experimentation and extensive statistical testing. A good example is DES where the substitution boxes have no publicly known structure. Therefore, one is left with a choice of either using costly lookup tables or a direct logic implementation. Although less regular and more difficult to design, the latter choice tends to be much more efficient. Recently proposed algorithms (e.g. AES, and the 3GPP Standard's Kasumi block cipher [13]) have substitution boxes that may be expressed as algebraic functions (e.g. as inversion in a binary finite field). This makes it possible to use the algebraic structure (e.g. use composite or tower field representation) to have more efficient substitution computations. Another good example is algebra on elliptic curves defined over optimal extension fields (e.g. $GF(p^k)$ where typically p fits into a word and is of pseudo-Mersenne form). Such arithmetic is commonly implemented in two levels: polynomial arithmetic to implement the operations in the field extension, and then $GF(p)$ arithmetic to support the coefficient operations.

In this context we pose the following question: Assume we are given a multivariate function (or polynomial) f of fixed degree defined over a finite ring Z_n (or field $GF(p)$) randomly picked[1] from the set of all such functions. What is the best choice for a parallel circuit implementation of f:

1. a binary logic realization of the function f (bit-level approach), or
2. a realization of f over Z_n (or $GF(p)$) (word-level approach)?

Intuitively, the later (word level) approach is attractive since it allows one to use the present mathematical structure in the implementation. However, since any circuit contructed from optimal binary logic implementations of word level function blocks can be re-optimized at the binary operator level as a whole, the potential of significant size advantage of the latter may overwhelm the design convenience of the former. In this paper we will obtain a measure of the relative merits of optimal binary versus optimal word level implementations for unstructured functions defined over a finite field or ring.

The remainder of this paper is as follows. We present a brief background on multivariate function (or polynomial) evaluation in the next section and introduce related notation and the multivariate formulation of Horner's algorithm. Then a generalization of Horner's algorithm to higher characteristic fields and rings is developed. We also present specialized optimization techniques based on the field/ring structure. The paper concludes by comparing the bit-level algorithm with the word-level approach in terms of bit-complexities.

[1] It is important to note that, the functions we use in cryptography are not always random and thus may permit more efficient computation. The work in this paper focuses on arbitrarily chosen functions and hence may be more suitable for the implementation of substitution boxes defined over finite rings or fields.

2 Background

The problem of function (or polynomial) evaluation has a rich history. The prominent polynomial evaluation algorithm attributed to Horner [3] has found its way into many applications due to its simplicity and efficiency. The univariate Horner's algorithm was shown by Ostrowski [9] to be optimal in the number of additions in the straight line algorithm in 1954. More than a decade later in 1966 Pan [10] proved its optimality in the number of multiplications as well. Furthermore, in [1] Borodin proved the uniqueness of Horner's algorithm, i.e. that all algorithms of similar complexity reduce to Horner's algorithm, as initially conjectured by Ostrowski.

Although Horner's algorithm is optimal for evaluating polynomials with arbitrary coefficients there are more efficient algorithms for evaluating polynomials by allowing precomputation on the coefficients. For example, the polynomial $p(x) = x^n$ can be computed by using only $\log_2 n$ multiplications and no additions. Similar algorithms can be derived for polynomials with less structure. In fact, when precomputation is allowed the evaluation can be achieved using only about $\frac{n}{2}$ multiplications [4].

Using the multivariate version of Horner's method it is possible to efficiently implement boolean functions. In an early result Shannon proved [12] a lower bound as $O(\frac{2^n}{n})$ on the size of circuits implementing arbitrary boolean functions of n variables. Optimally solving this problem, Muller gave an explicit construction based on a modification on Horner's method which satisfies the lower bound[5].

3 Preliminaries

We consider the problem of evaluating multivariate polynomials over Z_m using Horner's method. In the univariate case a polynomial of degree $r-1$ over Z_m is represented as

$$u(x) = u_0 + u_1 x + u_2 x^2 + \ldots + u_{r-1} x^{r-1} \quad , \quad u_i \in Z_m \ .$$

The most general polynomial one need consider is such that $r = order(Z_m)$ as any other polynomial may be reduced to this. A naive evaluation of $u(x)$ would require $r-1$ additions and $2r-3$ multiplications in Z_m. By the application of Horner's method, however, the following paranthesization is obtained.

$$u(x) = u_0 + x(u_1 + x(u_2 + x(u_3 + \ldots + x(u_{r-2} + xu_{r-1}))\ldots)$$

Now the polynomial can be evaluated by computing only $r-1$ additions and $r-1$ multiplications without using any temporary storage. A parallel implementation introduces a delay of

$$T = (r-1)T_A + (r-1)T_M$$

where T_A and T_M denote the delay of two input addition and multiplication operations in Z_m.

4 The Multivariate Case

A multivariate polynomial of n variables may be represented in sum of products representation as follows

$$u(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{r^n} u_i \prod_{j=1}^{n} x_j^{i_{j-1}}$$

where i_j denotes the j-th digit in the base r representation of i. Here again, the most general such polynomial involves each literal, x_j up to only the power $r-1$ as any other polynomial is reducible to this case. It is for this reason that a natural description of the "degree" of such polynomials is the maximum degree of any literal versus the commonly used measure of a polynomial's total degree. We set $u^{(1)}(x_1, x_2, \ldots, x_n) = u(x_1, x_2, \ldots, x_n)$ and expand it in powers of x_1 as follows

$$u^{(1)}(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{r-1} u^{(2)}(x_2, \ldots, x_n) x_1^i \ .$$

With the expansion we can now treat the summation as an $(r-1)$-st degree univariate polynomial of an indeterminate x_1. This enables us to use Horner's method as introduced earlier to evaluate the polynomial using $r-1$ additions and $r-1$ multiplications assuming the values of the coefficients are available. Note that the coefficients are still polynomials, however, the x_1 indeterminate is eliminated. The same expansion can now be applied on the r coefficients $u^{(2)}(x_2, \ldots, x_n)$ with x_2 as the indeterminate. We repeatedly expand the coefficients in the same fashion until polynomials in only x_n are obtained. The expansion will be repeated n times and in each step a level is obtained with one less variable in which the number of coefficients will grow by a factor of r as shown in Table 1. The process is recursively iterated as follows.

$$u^{(k)}(x_k, \ldots, x_n) = \sum_{i=0}^{r-1} u^{(k+1)}(x_{k+1}, \ldots, x_n) x_k^i \quad , \quad k = 1, 2, \ldots, n \ .$$

The total number of additions or multiplications is found as

$$C = \sum_{i=1}^{n} (r-1) r^{i-1} = r^n - 1 \ .$$

We summarize this result in the following theorem.

Theorem 1 (Multivariate Horner). *The evaluation of an n-variate polynomial over Z_m of maximum degree $(r-1)$ in all variables requires at most $r^n - 1$ additions and $r^n - 1$ multiplications in Z_m.*

A parallel implementation of n levels creates a total delay of

$$T = n(r-1)T_A + n(r-1)T_M \ .$$

Table 1. Number of coefficient polynomials introduced in each level

Level	#Coefficient Polynomials	#Mult or #Add
1	r	$(r-1)$
2	r^2	$(r-1)r$
3	r^3	$(r-1)r^2$
$\vdots$	$\vdots$	$\vdots$
n	r^n	$(r-1)r^{n-1}$

5 Our Contribution

We follow the same strategy as in Horner's algorithm, however, we make a key observation. In the last level in Table 1 the number of coefficients is given as r^n. These coefficients are univariate polynomials in x_n with maximum degree of $r-1$. However, the number of unique polynomials of this degree in Z_m is m^r. If $r^n > m^r$ then many of the coefficients which we counted as distinct are redundant. In the worst case the number of distinct coefficients for level n is therefore m^r. The same argument can be made for level $n-1$. In this level, the coefficients are polynomials in x_{n-1} with r coefficients that are polynomials in x_n. The number of unique polynomials in this level is therefore m^{2r}. The number of unique polynomials for each level is shown in Table 2. In the table we observe that the number of coefficient polynomials increases while the number of unique polynomials decreases with increasing levels. This suggests an optimization strategy which would compute the recursion using Horner's method until the number of coefficient polynomials exceeds the number of unique polynomials. Then for the remaining levels we compute *all* unique polynomials. Before continuing our analysis we find it instructive to illustrate our optimization strategy on a simple example:

Example 1. Let $Z_m = Z_2$ and $f = f(x_1, x_2, x_3, x_4)$ represent a multivariate polynomial $f : (Z_2)^4 \mapsto Z_2$ explicitly given as

$$f = x_1x_2x_3x_4 + x_1x_2x_3 + x_1x_2x_4 + x_2x_3x_4 + x_1x_3 + x_3x_4 + x_2x_4 + x_3x_4 + x_3 + x_2 + x_1 + 1 \ .$$

Applying Horner's algorithm we convert the polynomial into the following representation[2]

$$f = 1x_1\,[1x_2\{1x_3(1x_4+1)+(1x_4+0)\}+\{1x_3(0x_4+1)+(1x_4+1)\}] + [1x_2\{1x_3(1x_4+0)+(1x_4+1)\}+\{1x_3(1x_4+1)+(0x_4+1)\}]$$

We make out point by simply focusing on the last level of computation. Now note that in the last level we have 8 polynomial evaluations of the form $ax_4 + b$

[2] We use different kinds of parantheses to hint the computations that take place in each level.

where $a, b \in Z_2$. However, there can be only 2^2 such polynomials. Hence, a blind implementation of Horner's algorithm will be redundant. Since our algorithm is generic (and therefore should not depend on the particular choice of polynomial coefficients) in the last level we compute all possible polynomials in x_4, and simply wire the outputs to as many locations as required in the last level of the circuit evaluating f.

To compute all unique polynomials in level n, in which all polynomials are univariate over x_n, we use $r-1$ multiplications and $r-1$ additions per polynomial evaluation and $(r-1)m^r$ in total. Similarly in level $n-1$, all polynomials are now (since all polynomials over x_n are already computed) univariate over x_{n-1}. There are r coefficients with m^r choices for each coefficient. Hence there are $(m^r)^r = m^{r^2}$ polynomials requiring $(r-1)m^{r^2}$ additions and multiplications. This process is repeated until the first level is reached. The resulting complexities for each level are shown in Table 2.

To find the level k in which the number of coefficients exceeds the number of unique polynomials we need to find the smallest value of k satisfying the following inequality

$$r^k \geq m^{r^{n-k+1}} \ . \tag{1}$$

By taking the logarithm of both sides the following inequality is obtained

$$kr^k \geq r^{n+1} \log_r m \ . \tag{2}$$

Let the right-hand-side be called c. Taking the logarithm of both sides with respect to base r and solving for equality we obtain

$$k = \log_r c - \log_r k \ .$$

Now we may substitute the value of k in the logarithm on the right-hand-side.

$$k = \log_r c - \log_r(\log_r c - \log_r k) \ .$$

We may continue in the same fashion substituting infinitely many times.

$$k = \log_r c - \log_r(\log_r c - \log_r k(\log_r c - \log_r k(\log_r c - \log_r(\ldots)\ldots) \ .$$

Table 2. Number of coefficient polnomials and unique polynomials at each level

Level	#Coefficient Polynomials	#Mult or #Add	#Unique Polynomials	#Mult or #Add
1	r	$(r-1)$	m^{nr}	$(r-1)m^{r^n}$
2	r^2	$(r-1)r$	$m^{(n-1)r}$	$(r-1)m^{r^{n-1}}$
3	r^3	$(r-1)r^2$	$m^{(n-2)r}$	$(r-1)m^{r^{n-2}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-2$	r^{n-2}	$(r-1)r^{n-3}$	m^{3r}	$(r-1)m^{r^3}$
$n-1$	r^{n-1}	$(r-1)r^{n-2}$	m^{2r}	$(r-1)m^{r^2}$
n	r^n	$(r-1)r^{n-1}$	m^r	$(r-1)m^r$

Note that by each new term the value of k becomes more precise. At the same time the contribution of these terms shrink logarithmically. Since we are interested in only integer values of k it suffices to approximate k by neglecting the terms after only two levels of substitution.

$$k \approx \log_r c - \log_r(\log_r c) \ . \tag{3}$$

The exact solution of (2) is defined in terms of the Lambert-W function [2]

$$k \geq W(\log r \frac{r^{n+1}}{\log_m r}) / \log r \tag{4}$$

where $W(x)$ is defined as the inverse of the map $x \rightarrow xe^x$.

Now we can compute the total number of operations by simply summing the entries in the third column from the first level through level k and the entries in the last column from level $k+1$ through level n in Table 2.

$$\begin{aligned} C &= \sum_{i=1}^{k}(r-1)r^{i-1} + \sum_{i=1}^{n-k}(r-1)m^{r^i} \\ &= (r^k - 1) + (r-1)(m^r + m^{r^2} + m^{r^3} + \ldots + m^{r^{n-k}}). \end{aligned} \tag{5}$$

Ignoring the smaller order terms in the super-exponential summation the complexity may be approximated as follows

$$C \approx r^k + rm^{r^{n-k}} \ .$$

Using (3) and r^{n-k} directly obtained from (2) the complexity is further simplified as

$$\begin{aligned} C &= \frac{c}{\log_r c} + rm^{\frac{n \log_m r}{r}} \\ &= \frac{r^{n+1} \log_r m}{(n+1) + \log_r(\log_r m)} + r^{\frac{n}{r}+1} \end{aligned} \tag{6}$$

Hence, both the addition and multiplication complexities grow by $O(\frac{r^n}{n})$.

Theorem 2 (Modified Horner). *Given $r^n > m^r$ the evaluation of an n-variate polynomial over Z_m of maximum degree $n(r-1)$ requires at most $O(\frac{r^n}{n})$ additions and multiplications in Z_m.*

In the improved algorithm a parallel implementation of n levels creates a total delay of

$$T = k(r-1)(T_A + T_M) + (n-k)(r-1)(T_{A,const} + T_{M,const}) \ .$$

Here $T_{A,const}$ and $T_{M,const}$ denote delays of *constant* addition and multiplication in Z_m, respectively.

The structure that results is depicted in Figure 1. The product and the summation symbols indicate blocks implementing multiplication and addition in Z_m, respectively. Note that, in each level up to the $k-1$ level the fan-out of the polynomial implementing arithmetic blocks is unity while for higher levels (i.e. $k, k+1, \ldots, n$) the fan-out may be greater than one.

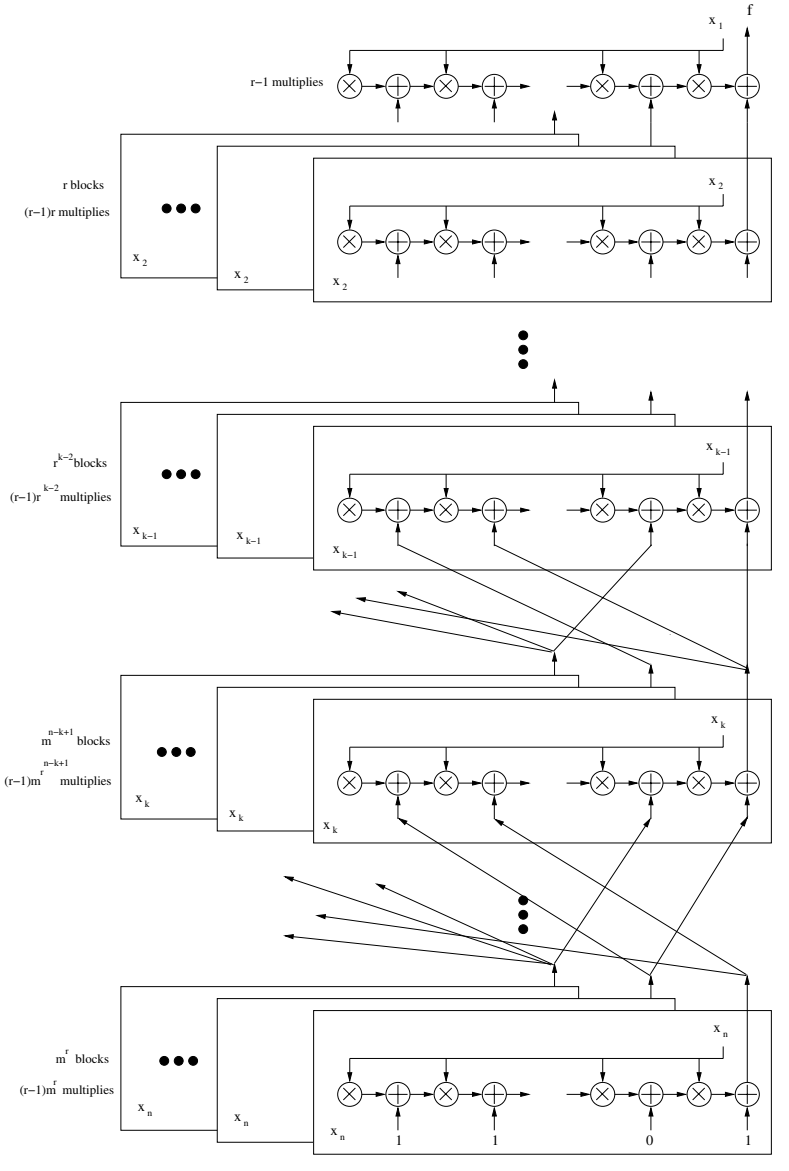

Fig. 1. Block diagram of a generic circuit implementing the modified Horner algorithm

6 Polynomials over Prime Fields

For polynomials over prime fields $GF(p)$ the polynomial degree is bounded by $p-1$ as any polynomial of higher degree may be reduced using Fermat's Theorem:

$x^{p-1} = 1 \bmod p$. Substituting $m = p$ and $r = p$ in (6) we obtain the total number of additions and multiplications as [3]

$$C \approx \frac{p^{n+1}}{n+1} \tag{7}$$

Hence, both the addition and multiplication complexities grow by $O(\frac{p^n}{n})$. On the other hand, using $k = (n+1) - \log_p(n+1)$ derived from (3) the time complexity simplifies as follows

$$T = ((n+1)-\log_p(n+1))(p-1)(T_A+T_M)+(\log_p(n+1)-1)(p-1)(T_{A,const}+T_{M,const}) .$$

Theorem 3 (Modified Horner over $GF(p)$). *Given $n > p$ the evaluation of an n-variate polynomial over $GF(p)$ requires at most $O(\frac{p^n}{n})$ additions and multiplications in $GF(p)$ with a delay of $O((p-1)(n-\log_p n))$.*

7 Optimality

Consider a parallel implementation of a function in $GF(p)$ of size s which denotes the total number of addition and multiplication components used in the circuit. The total number of circuits that can be built using s components is

$$(((s+p+n)^2)^s)$$

since each of the s components can have either the output of another component (s choices) a constant (p choices) or a literal (n choices) connected as input. To build an arbitary function, the number of circuits must exceed the number of n-variate functions over $GF(p)$. This leads to the following inequality.

$$((s+p+n)^2)^s \geq p^{p^n}$$

Substituting $s = \frac{p^n}{2n}$ we see that (although close) the inequality is still not satisfied

$$(\frac{p^n}{2n}+p+n)^{\frac{p^n}{2n}} \approx \frac{p^{p^n}}{2n} \not\geq p^{p^n}$$

Hence we found a lower bound on the circuit complexity any circuit must satisfy to evaluate an arbitrary n-variate function over $GF(p)$.

Theorem 4 (Lower Bound on Circuit Size). *Any circuit evaluating an arbitrary n-variate polynomial over $GF(p)$ requires at least $\Omega(\frac{p^n}{2n})$ adders and multipliers in $GF(p)$.*

The bound can be made tighter by more careful analysis but if suffices for our purposes. With Theorems 3 and 4 it directly follows that the presented modification to Horner's algorithm for multivariate polynomials over $GF(p)$ is asymptotically optimal.

[3] In the $p = 2$ case, the multiplications in the second summation disappear since they are constant multiplications by either 0 or 1.

8 Comparison to Muller's Algorithm

The Muller construction [5] gives a method for evaluating arbitrary n-variate polynomials over $GF(2)$ with $O(\frac{2^{n+1}}{n+1})$ complexity (see (7)). We may consider the task of evaluating an n-variate polynomial over $GF(p)$ as equivalent to evaluating $(\log_2 p)$ polynomials of $(n \log_2 p)$-variables over $GF(2)$. In this case the bit-level algorithm implementing a polynomial evaluation over $GF(p)$ has bit-complexity

$$C_B = O\left((\log_2 p)\frac{2^{n \log_2 p+1}}{n \log_2 p + 1}\right) = O\left(\frac{2p^n}{n}\right) .$$

On the other hand the complexity equation (7) derived in this paper may be expressed in bit operations rather than operations in $GF(p)$. Then assuming a $GF(p)$ multiplication operation takes $(\log_2 p)^2$ bit operations we obtain the bit complexity as follows

$$C_W = O\left(\frac{p^{n+1}}{n+1}(\log_2 p)^2\right) .$$

Interestingly, the bit-level algorithm seems to be a constant $\frac{p}{2}(\log_2 p)^2$ times more area efficient[4]. Note that in the $GF(p)$ case we are limiting our algorithms to operate on groups of $\log_2 p$ bits whereas Muller's algorithm operates on individual bits. Due to the fine grained approach Muller's algorithm has more opportunity for optimization.

We see a similar picture in the time complexities. We may assume both the $GF(p)$ multiplication and the addition circuits compute the result in $O(\log_2 \log_2 p)$ two-input gate delays where $p > 2$ using a fast addition circuit. Thus ignoring the constant operations the overall computation takes

$$T_W = O((p-1)(\log_2 \log_2 p)(n - \log_p n)).$$

gate delays in the word-level approach. The bit-level approach yields a time complexity of

$$T_B = O(n \log_2 p - \log_2(n \log_2 p)).$$

gate delays. The bit-level algorithm seems to be roughly $\frac{(p-1)(\log_2 \log_2 p)}{\log_2 p}$ times faster than the $GF(p)$ algorithm.

9 Further Optimizations

Up until now we have not used any special properties of the ring structure. One strategy that comes to mind is to use Euler's Theorem to reduce the polynomial degree r. For relatively prime integers a and m Euler's Theorem is simply stated as $a^{\phi(m)} = 1 \pmod m$. When the degree r of $u(x)$ is larger than $\phi(m)$, then by restricting $x_1, x_2, \ldots, x_n$ to integers that are relatively prime to m we obtain the following strategy:

[4] This figure may be reduced by employing fast (FFT based) methods to realize $GF(p)$ multiplications.

– First compute

$$u'(x_1, x_2, \ldots, x_n) = u(x_1, x_2, \ldots, x_n) \bmod (x_1^{\phi(m)}, x_2^{\phi(m)}, \ldots, x_n^{\phi(m)})$$

offline.
– Evaluate $u(x_1, x_2, \ldots, x_n)$ by evaluating $u'(x_1, x_2, \ldots, x_n)$.

With this strategy the direct application of the modified Horner's algorithm has complexity $O(\frac{\phi(m)^n}{n})$.

It is possible to obtain further improvements by using the factorization of the modulus m to use efficient residue arithmetic. For instance, if m is factorized into distinct prime powers as $m = p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}$, then we may achieve the evaluation in two steps:

– Evaluate $u(x)$ with respect to moduli $p_1^{e_1}, p_2^{e_2}, \ldots, p_t^{e_t}$.
– Use the Chinese Remainder Theorem (CRT) to assemble the result w.r.t. modulus m.

Note that this evaluation procedure may provide more than the standard speedup obtained by the CRT. If we know that $x_1, x_2, \ldots, x_n$ are not divisible by any prime factor of m then the t polynomial evaluations may be performed by evaluating $u'(x) = u(x) \bmod x^{\phi(p_i^{e_i})}$ for $i = 1, 2, \ldots, t$. Then the total complexity becomes

$$C = \frac{1}{n} \sum_{i=1..t} \phi(p_i^{e_i})^n = \frac{1}{n} \sum_{i=1..t} (p_i^{e_i} - p_i^{e_i - 1})^n$$

To gain more insight we assume roughly equal sized partitions, i.e. $p_i^{e_i} \approx m/t$ and simplify the complexity further to

$$C \approx \frac{1}{n} \sum_{i=1..t} (m/t)^n = \frac{m^n}{n t^{n-1}}$$

Note that this complexity figure gives the number of addition and multiplication operations carried out in rings roughly of size m/t which is much smaller in size than Z_m. Hence there is additional improvement in the bit-complexities. Nevertheless, we observe that the complexity $O(\frac{m^n}{nt^{n-1}})$ is exponentially improved by growing t. The complexity of residue computations and the CRT re-construction are not included in this partial result. The complexity of these additional operations is a strong function of the prime power decomposition of the modulus. However, for a large modulus the result, i.e. C, is expected to dominate the overall complexity.

10 Conclusion

We presented a means of improving the parallel implementation complexity of evaluating unstructed n-variate polynomials over finite rings and fields. Our modification is based on eliminating redundancies in the multivariate version

of Horner's algorithm which occur when the evaluation takes place over a small finite mathematical structure and may be considered as a generalization of Shannon's lower bound and Muller's algorithm to word level circuits.

We presented two strategies for further improving the multivariate version of Horner's algorithm which utilize the ring structure by employing residue arithmetic via the Chinese Remainder Theorem. It turns out that by restricting the inputs to integers relatively prime to m, exponential improvement can be obtained. Of course, this statement is based on the assumption that m is highly composite.

If the domain is a finite field $GF(p)$ the complexity of multivariate Horner polynomial evaluation is improved from $O(p^n)$ to $O(\frac{p^n}{2n})$. We prove the optimality of the presented algorithm and show that the bit-level algorithm provides a constant times better time and space complexities than the word-level approach. The lesson taught by this exercise is that the currently popular implementation style which favors the word-level approach diverges from optimality as the order of the finite field increases. We have shown that the bit-level approach provides significant time and area savings provided that the function is chosen arbitrarily, which is the case for substitution boxes in cryptographic applications. We should point out that our result will not apply to highly structured specialized functions since there is significantly more opportunity for optimization by using the special structure of the function.

References

1. A. Borodin. Horners Rule is Uniquely Optimal. In Z. Kohavi and A. Paz, editors, *Proceedings of an International Symposium on the Theory of Machines and Computations*, pages 45–57. Academic Press, 1971.
2. R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W Function. *Advances in Computational Mathematics*, 5:329–359, 1996.
3. W. G. Horner. A new method of solving numerical equations of all orders by continuous approximation. *Philos. Trans. Roy. Soc. London*, 109:308–335, 1819.
4. D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, USA, 2nd edition, 1981.
5. D. E. Muller. Complexity in Electronic Switching Circuits. *IRE Transactions on Electronic Circuits*, (5):15–19, 1956.
6. NIST FIPS PUB 180-1. *Secure Hash Standard*. Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce, April 1995.
7. NIST FIPS PUB 46-3. *Data Encryption Standard*. Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce, 1977.
8. U.S. Department of Commerce/National Institute of Standard and Technology. *Advanced Encryption Standard (AES)*, November 2001.
9. A. M. Ostrowski. On two problems in abstract algebra connected with Horner's rule. pages 40–48. Academic Press, 1954. presented to Richard von Mises.
10. V. Ya. Pan. Methods for Computing Values of Polynomials. *Russian Mathematical Surveys*, 21(1):105–136, 1966.

11. R.L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, April 1992.
12. C. E. Shannon. The Synthesis of Two-terminal Switching Circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.
13. ETSI/SAGE Specification. Specification of the 3GPP confidentiality and integrity algorithms; part 2: KASUMI specification. 3GPP TS 35.202, European Telecommunications Standards Institute, Sophia-Antipolis Cedex, France, November 1999. Draft.