The Complexity and Parallel Implementation of two Sparse Multivariate Hensel Lifting Algorithms for Polynomial Factorization

Tian Chen and Michael Monagan

Department of Mathematics, Simon Fraser University,
Burnaby, British Columbia, V5A 1S6, CANADA
tca71@sfu.ca, mmonagan@cecm.sfu.ca

Abstract. Sparse multivariate Hensel lifting (SHL) algorithms are used in multivariate polynomial factorization. They improve on Wang's classical multivariate Hensel lifting which can be exponential in the number of variables for sparse factors.

In this work, we present worst case complexity analyses and failure probability bounds for two recently developed SHL algorithms. One of the algorithms solves the multivariate Diophantine equations using sparse interpolation, and the other interpolates the factors directly from bivariate images obtained using bivariate Hensel lifting.

We have observed that a linear expression swell occurs in both approaches. We have modified the second approach to eliminate the expression swell. Our improvement also injects more parallelism into the sparse interpolation step.

We have made a high-performance parallel implementation of our SHL algorithm in Cilk C. We present timing benchmarks comparing our Cilk C implementation with the factorization algorithms in Maple and Magma. We obtain good parallel speedup and our algorithm is much faster than Maple and Magma on our benchmarks.

Keywords: Sparse Multivariate Hensel Lifting, Sparse Interpolation, Multivariate Diophantine Equations, Polynomial Factorization, Bivariate Hensel Lifting, Cilk C

1 Introduction

Polynomial factorization has been a central topic in computer algebra, and it continues to play a critical role in other fields such as algebraic coding theory, cryptography, number theory and algebraic geometry [5]. In this work, we focus on the main tool used to factor multivariate polynomials, namely, multivariate Hensel lifting (MHL). MHL was initially developed by Yun [19] and Wang [18] to factor polynomials with integer coefficients, but it can be applied to polynomials with coefficients in other domains, for example, finite fields [1, 4] and algebraic number fields [15, 16, 23].

To factor a multivariate polynomial $a \in \mathbb{Z}[x_1, x_2, \cdots, x_n]$, Wang's multivariate Hensel lifting [18] first chooses integers $\alpha_2, \ldots, \alpha_n$ and factors the univariate

image $a(x_1, \alpha_2, \ldots, \alpha_n)$ in $\mathbb{Z}[x_1]$. Then it recovers the multivariate factors from their images one variable at a time. A key step in Wang's MHL is the solution of a sequence of multivariate polynomial diophantine equations (MDPs). Wang's MHL has been implemented in many computer algebra systems including Maple, Magma, Macsyma, Mathematica and Singular. For a detailed description of Wang's MHL we refer the reader to Chapter 6 of [5].

It is known that when factors are sparse and the evaluation points $\alpha_2, \ldots, \alpha_n$ are mostly non-zero, Wang's method for solving MDPs can be exponential in the number of variables [9, 12]. To resolve this, Zippel [21] introduced the first polynomial-time probabilistic algorithm in 1981 that takes advantage of sparsity. Other sparse Hensel lifting (SHL) algorithms were developed by Kaltofen in 1985 [6] and Kaltofen and Trager in 1990 [7].

In 2016, Monagan and Tuncer [9] proposed a new sparse Hensel lifting algorithm called MTSHL. The authors made a key observation which they call the strong SHL assumption (see Lemma 1 of [9]) which is applied to solve the MDPs that appear in Wang's MHL in random polynomial time. A detailed complexity analysis for MTSHL was completed for the average-case in [12]. MTSHL was integrated into Maple 2019 [13].

In 2018, Monagan and Tuncer [11] introduced another approach [11] that does not solve MDPs. Instead, at each Hensel lifting step, it interpolates the factors from many bivariate images which are obtained using bivariate Hensel lifting. Classical bivariate Hensel lifting (BHL) costs $O(d^4)$ where $d = \deg(a)$ is the total degree of the input polynomial. The cost of BHL is improved to $O(d^3)$ by Monagan in [14]. This approach is appropriate for multivariate Hensel lifting because the degree of the factors is rarely 100, and often 10 or lower.

Our work is motivated by the following observation. In the main Hensel lifting step (see Algorithm 1) which is used in MTSHL ([12]), in [11] and also in Wang's multivariate Hensel lifting [18], when the evaluation point α_j is non-zero, an expression swell occurs in each factor as it is recovered (in line 13). This increases the cost of the error computation (in line 14).

Our first contribution is a new algorithm CMSHL which reorganizes the sparse Hensel lifting algorithm in [11] to eliminate the expression swell. Our second contribution is a worst case complexity analysis for CMSHL and for MTSHL and bounds for the failure probability of both algorithms. Our third contribution is a high-performance parallel implementation of CMSHL using Cilk C [3] for multi-core computers.

Our paper is organized as follows. In Section 2, we present the two sparse multivariate Hensel lifting algorithms from [9, 12] and [11]. We study the expression swell and give examples of it for the worst case and then we present our new algorithm CMSHL which eliminates the expression swell. In Section 3 we give worst case complexity analyses for MTSHL and CMSHL along with their failure probabilities. In Section 4 we present timings comparing our Cilk C implementation of CMSHL with Maple and Magma's factorization commands for a variety of input problems. The timings (see Tables 3, 4 and 5) demonstrate good parallel speedup. In Section 5 we give some details of our Cilk C implementation.

2 Two algorithms of sparse multivariate Hensel lifting

Suppose we seek the factors of a multivariate polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$. Similar to Wang's multivariate Hensel lifting (MHL), a few preliminary steps are done before sparse multivariate Hensel lifting (SHL) [10]:

The first step is to compute and remove the content of a in a chosen main variable, say x_1. For $a = \sum_{i=0}^{d} a_i(x_2, \cdots, x_n)x_1^i$, the content of a is $\gcd(a_0, \cdots, a_d)$, a polynomial with one fewer variables which can be factored recursively. Let us assume this has already been done.

The second step is to identify any repeated factors in a by doing a *square-free factorization* (see ch.8 in [5]). After this, we obtain the factorization $a = b_1 b_2^2 \cdots b_k^k$ such that each factor b_i is square-free and $\gcd(b_i, b_j) = 1$ for $i \neq j$. Suppose this has also been done and let $a = f_1 f_2 \cdots f_r$ be the irreducible factorization of a over $\mathbb{Z}$.

Next, an evaluation point $\alpha = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ is chosen and then $a(x_1, \alpha)$ is factored over $\mathbb{Z}$. The evaluation point α must satisfy the following conditions: (i) $L(\alpha) \neq 0$ where L is the leading coefficient of a in x_1, (ii) $a(x_1, \alpha)$ must have no repeated factors in x_1, and (iii) $f_i(x_1, \alpha)$ must be irreducible. Conditions (i) and (ii) can be enforced in advance whereas (iii) can be ensured with high probability by choosing the integers α_i from a sufficiently large set.

For simplicity, throughout this paper we only consider two irreducible factors f and g both monic in x_1. For multi-factor cases, we refer the reader to [10]. Let $a = fg$ where f and g are monic irreducible polynomials in $\mathbb{Z}[x_1, \cdots, x_n]$. We define $h_j := h(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n)$ for a polynomial $h \in \mathbb{Z}[x_1, \cdots, x_n]$. We use the notation $\#h$ to be the number of non-zero terms of h. To factor a, the image a_1 is first factored over $\mathbb{Z}$. From Hilbert's irreducibility theorem (see e.g. [8]), $f(x_1, \alpha)$ and $g(x_1, \alpha)$ are irreducible with high probability.

Now we start the process of sparse multivariate Hensel lifting to recover f and g from a, f_1, g_1. The inputs are a, f_1, g_1, α and a prime p such that $\gcd(f_1, g_1) = 1$ in $\mathbb{Z}_p[x_1]$. The algorithm lifts (f_1, g_1) to (f_2, g_2), then lifts (f_2, g_2) to (f_3, g_3) etc. until (f_n, g_n) is obtained. At each step, $a_j - f_j g_j \bmod p = 0$ so that at the final step, $a_n - f_n g_n \bmod p = 0$. To recover the integer coefficients in the factors one may either use a sufficiently large p or do a subsequent p-adic lift [10].

2.1 MTSHL and CMSHL

The j^{th} Hensel lifting step for both approaches of sparse multivariate Hensel lifting in [9, 12] and [11] is presented. Our presentation includes worst case complexity bounds for the main steps as an aid for the reader and for later reference.

The first approach (MTSHL [9, 12]) is presented in Algorithms 1 and 2. Algorithm 2 is called from Algorithm 1 in a loop to solve the MDPs via sparse interpolation. Note that in Algorithm 2, if $\max(t_i)$ is much larger (or much smaller) than $\max(s_i)$ then it will be faster to interpolate the smaller of σ and τ only and obtain the larger of σ and τ using $\sigma u + \tau w = c$. The second approach in [11] is shown in Algorithm 3.

Algorithm 1 MTSHL: Hensel lift x_j with MDPs via sparse interpolation.

Input: A prime p, $\alpha_j \in \mathbb{Z}_p$, $a_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ monic in x_1, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ s.t. $a_j(x_1, ..., x_{j-1}, \alpha_j) = f_{j-1}g_{j-1}$ with $j > 2$.
Output: $f_j, g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ s.t. $a_j = f_j g_j$ where $f_j(x_j = \alpha_j) = f_{j-1}$ and $g_j(x_j = \alpha_j) = g_{j-1}$; Otherwise, **return** FAIL.
$(\sigma_0, \tau_0) \leftarrow (f_{j-1}, g_{j-1})$; $(f_j, g_j) \leftarrow (f_{j-1}, g_{j-1})$.
error $\leftarrow a_j - f_j g_j$; monomial $\leftarrow 1$.
for $i = 1, 2, \cdots$ **while** error $\neq 0$ and $\deg(f_j, x_j) + \deg(g_j, x_j) < \deg(a_j, x_j)$ **do**
  monomial $\leftarrow$ monomial $\cdot\ (x_j - \alpha_j)$.
  $c_i \leftarrow \text{coeff}(\text{error}, (x_j - \alpha_j)^i)$.
  if $c_i \neq 0$ **then**
    // Solve the MDP $\sigma_i g_{j-1} + \tau_i f_{j-1} = c_i$ for $\sigma_i, \tau_i \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$.
    $\sigma_f \leftarrow \sigma_{i-1}$; $\tau_f \leftarrow \tau_{i-1}$.
    $(\sigma_i, \tau_i) \leftarrow \text{SparseInterp}(g_{j-1}, f_{j-1}, c_i, \sigma_f, \tau_f)$
    if $(\sigma_i, \tau_i) = \text{FAIL}$ **then return** FAIL(1) **end if**
    $(f_j, g_j) \leftarrow (f_j + \sigma_i \cdot \text{monomial}, g_j + \tau_i \cdot \text{monomial})$.
    error $\leftarrow a_j - f_j g_j$.
  end if
end for
if error $= 0$ **then return** (f_j, g_j) **else return** FAIL(2) **end if**

Algorithm 2 SparseInterp: solve an MDP using sparse interpolation.

Input: $u, w, c, \sigma_f, \tau_f \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ where u, w are monic in x_1.
Output: The solution (σ, τ) to the MDP $\sigma u + \tau w = c \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ or FAIL.
Let $d\sigma = \deg(\sigma_f, x_1)$ and $\sigma = \sum_{i=0}^{d\sigma} \zeta_i(x_2, \cdots, x_{j-1}) x_1^i$ with $\zeta_i = \sum_{l=1}^{s_i} a_{il} M_{il}$, and let $d\tau = \deg(\tau_f, x_1)$ and $\tau = \sum_{i=0}^{d\tau} \eta_i(x_2, \cdots, x_{j-1}) x_1^i$ with $\eta_i = \sum_{l=1}^{t_i} b_{il} N_{il}$, where a_{il}, b_{il} are to be determined and $x_1^i M_{il}$, $x_1^i N_{il}$ are monomials in σ_f, τ_f respectively.
Let s be the maximum of s_i and t_i.
Pick $(\beta_2, \cdots, \beta_{j-1}) \in (\mathbb{Z}_p \backslash \{0\})^{j-2}$ at random.
Compute monomial evaluations ..... $\mathcal{O}((j-2))(\#f + \#g + d_{\max}))$
$\mathcal{S} = \{S_i = \{m_{il} = M_{il}(\beta_2, \cdots, \beta_{j-1}) : 1 \leq l \leq s_i\}, 0 \leq i \leq d\sigma\}$ and
$\mathcal{T} = \{T_i = \{n_{il} = N_{il}(\beta_2, \cdots, \beta_{j-1}) : 1 \leq l \leq t_i\}, 0 \leq i \leq d\tau\}$.
if any $|S_i| \neq s_i$ **or** $|T_i| \neq t_i$ **then return** FAIL(1) **end if**
for k from 1 to s in parallel **do**
  Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
  Evaluate $u(x_1, Y_k), w(x_1, Y_k), c(x_1, Y_k)$. ..... $\mathcal{O}(s(\#f + \#g + \#a))$
  if $\gcd(u(x_1, Y_k), w(x_1, Y_k)) \neq 1$ **then return** FAIL(2) **end if**
  Solve $\sigma_k(x_1)u(x_1, Y_k) + \tau_k(x_1)w(x_1, Y_k) = c(x_1, Y_k) \in \mathbb{Z}_p[x_1]$. ..... $\mathcal{O}(s\, d_1^2)$
end for
for i from 0 to $d\sigma$ in parallel **do**
  Construct and solve the $s_i \times s_i$ linear system for a_{il}: ..... $\mathcal{O}(s\#f)$

$$\left\{ \sum_{l=1}^{s_i} a_{il} m_{il}^k = \text{coeff}(\sigma_k(x_1), x_1^i) \text{ for } 1 \leq k \leq s_i \right\}.$$

end for
Substitute the solution a_{il} into σ.
Similarly, construct τ. ..... $\mathcal{O}(s\#g)$
if $\sigma u + \tau w = c$ **then return** (σ, τ) **else return** FAIL(3) // wrong σ_f or τ_f

Algorithm 3 Hensel lift x_j via bivariate Hensel lifting [11].

1: **Input:** A prime p, $\alpha_j \in \mathbb{Z}_p$, $a_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ monic in x_1, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ s.t. $a_j(x_1, ..., x_{j-1}, \alpha_j) = f_{j-1}g_{j-1}$ with $j > 2$.
2: **Output:** $f_j, g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ s.t. $a_j = f_j g_j$ where $f_j(x_j = \alpha_j) = f_{j-1}$ and $g_j(x_j = \alpha_j) = g_{j-1}$; Otherwise, **return** FAIL.
3: Let $f_j = \sum_{h=0}^{df_j} \sigma_h(x_1, ..., x_{j-1})(x_j - \alpha_j)^h$ with $\sigma_h = \sum_{i=0}^{df} \left(\sum_{l=1}^{s_i} c_{hil} M_{il}\right) x_1^i$, where $M_{il} x_1^i$ are monomials in $\sigma_0 = f_{j-1}$ and $df = \deg(f_{j-1}, x_1)$. $df_j = \deg(f_j, x_j)$ TBD.
Let $g_j = \sum_{h=0}^{dg_j} \tau_h(x_1, ..., x_{j-1})(x_j - \alpha_j)^h$ with $\tau_h = \sum_{i=0}^{dg} \left(\sum_{l=1}^{t_i} d_{hil} N_{il}\right) x_1^i$, where $N_{il} x_1^i$ are monomials in $\tau_0 = g_{j-1}$ and $dg = \deg(g_{j-1}, x_1)$. $dg_j = \deg(g_j, x_j)$ TBD.
4: Pick $(\beta_2, \cdots, \beta_{j-1}) \in \mathbb{Z}_p^{j-2}$ at random.
5: Compute monomial evaluation sets ..... $\mathcal{O}((j-2)(\#f + \#g + d_{\max}))$
$\mathcal{S} = \{S_i = \{m_{il} = M_{il}(\beta_2, \cdots, \beta_{j-1}), 1 \le l \le s_i\}, 0 \le i \le df - 1\}$ and
$\mathcal{T} = \{T_i = \{n_{il} = N_{il}(\beta_2, \cdots, \beta_{j-1}), 1 \le l \le t_i\}, 0 \le i \le dg - 1\}$.
6: **if** any $|S_i| \neq s_i$ **or** any $|T_i| \neq t_i$ **then return** FAIL(1) **end if**
7: Let s be the maximum of s_i and t_i.
8: **for** k from 1 to s **in parallel do**
9: Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
10: $A_k, F_k, G_k \leftarrow a_j(x_1, Y_k, x_j), f_{j-1}(x_1, Y_k), g_{j-1}(x_1, Y_k)$. .. $\mathcal{O}(s(\#f + \#g + \#a))$
11: **if** $\gcd(F_k, G_k) \neq 1$ **then return** FAIL(2) **end if** // unlucky evalution
12: Call *BivariateHenselLift*$(A_k, F_k, G_k, \alpha_j, p)$ to compute $\sigma_{hk}(x_1)$ and $\tau_{hk}(x_1)$ s.t. $A_k = f_k g_k$ where $f_k = \sum_{h=0}^{df_j} \sigma_{hk}(x_j - \alpha_j)^h$ and $g_k = \sum_{h=0}^{dg_j} \tau_{hk}(x_j - \alpha_j)^h$.
13: **end for**
14: **for** h from 1 to df_j **do**
15: **for** i from 0 to df **do**
16: Construct and solve the $s_i \times s_i$ linear system for c_{hil} ..... $\mathcal{O}(sd_j \#f)$

$$\left\{ \sum_{l=1}^{s_i} c_{hil} m_{il}^k = \text{coeff}(\sigma_{hk}(x_1), x_1^i) \text{ for } 1 \le k \le s_i \right\}$$

17: **end for**
18: **end for**
19: Substitute the solution c_{hil} into σ_h and expand to get f_j. ..... $\mathcal{O}(d_j^2 \#f)$
20: Similarly to construct g_j. ..... $\mathcal{O}(sd_j \#g)$
21: **if** $a_j = f_j g_j$ **then return** (f_j, g_j) **else return** FAIL(3) **end if**

2.2 Intermediate expression swell

In Algorithm 1, an expression swell may occur in line 13 and 14. In Algorithm 3 an expression swell may occur at the final expansion step (line 19). To illustrate the expression swell, we consider the partial sums of f_j. Let

$$f_j^{(i)} = \sum_{k=0}^{i} \sigma_k(x_1, \cdots, x_{j-1})(x_j - \alpha_j)^k \text{ for } 0 \le i \le d_j, \text{ and}$$
$$f_{jH}^{(i)} = (((\sigma_{d_j}(x_j - \alpha_j) + \sigma_{d_j - 1})(x_j - \alpha_j) + \cdots)(x_j - \alpha_j)) + \sigma_{d_j - i}$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\#\sigma_i$	925	737	584	459	352	268	196	134	94	64	48	24	13	7	3
$\#f_j^{(i)}$	925	1512	1851	1999	**2021**	1934	1768	1628	1486	1411	1226	1130	1071	1028	989
$\#f_{jH}^{(i)}$	3	10	23	47	95	159	253	387	583	851	1203	1662	2246	**2983**	989

Table 1. Number of terms in $f_j^{(i)}$ and $f_{jH}^{(i)}$ with a randomly generated polynomial.

i	0	1	2	3	4
$\#\sigma_i$	7	7	7	7	7
$\#f_j^{(i)}$	7	14	21	28	7
$\#f_{jH}^{(i)}$	7	14	21	28	7

Table 2. Number of terms in $f_j^{(i)}$ and $f_{jH}^{(i)}$ in a worst case.

where $f_{jH}^{(i)}$ is the expansion in Horner's form and $d_j = \deg(f_j, x_j)$. $f_j^{(i)}$ and $f_{jH}^{(i)}$ correspond to the intermediate steps in line 13 of Algorithm 1 and in line 19 of Algorithm 3 respectively. We are interested in $\#f_j^{(i)}$ and $\#f_{jH}^{(i)}$ for $0 \leq i \leq d_j$.

Table 1 shows an example of a randomly generated polynomial with $p = 2^{31} - 1$, $j = 5$, $d_j = 14$, $d = 20$ and $\#f_j = 989$. The density ratio $\#f_j / \binom{d+j}{j} \approx 0.0186$. The ratios $\max(\#f_j^{(i)})/\#f_j$ and $\max(\#f_{jH}^{(i)})/\#f_j$ are 2.043 and 3.016 respectively. This example shows a typical trend in an average case where $\max(\#f_j^{(i)})/\#f_j \lessapprox 1 + d/j$ [12]. We observe that $\#\sigma_i$ decreases as i increases from 0 to d_j. $\#f_j^{(i)}$ increases to a peak in the first few expansions and gradually shrinks back to $\#f_j$. $\#f_{jH}^{(i)}$ increases to a higher peak than $\max(\#f_j^{(i)})$ and drops down to $\#f_j$ at the last iteration.

The following example illustrates the worst case where $\#f_j^{(i)}$ increases linearly to its maximum, $d_j \#f_j$.

$$f_j = (31x_3 + 100x_3^2 + (49 + 36x_2^2 + (x_1^4 + 44x_1^2 + 28)x_2^3)x_3^3)x_4^4,$$

with $p = 101$, $j = 4$ and $\#f_j = 7$. Table 2 shows the number of terms in $f_j^{(i)}$. $\max(\#f_j^{(i)})$ equals to $d_j \#f_j$. This is because

$$\sigma_i = \frac{1}{i!} \frac{\partial^{(i)} f_j}{\partial x_j^i} (x_j = \alpha_j) \text{ for } 0 \leq i \leq d_j$$

and in this example f_j only contains the terms with $x_j^{d_j}$. In this case, $\#\sigma_i$ is never reduced as i increases from 0 to d_j and we have $\max(\#f_j^{(i)})/\#f_j = d_j$.

2.3 Our new algorithm: CMSHL

We present a new approach which eliminates the expression swell in Algorithm 3. The idea is depicted in Fig. 1. Consider one of the factors f_j at the j^{th} Hensel

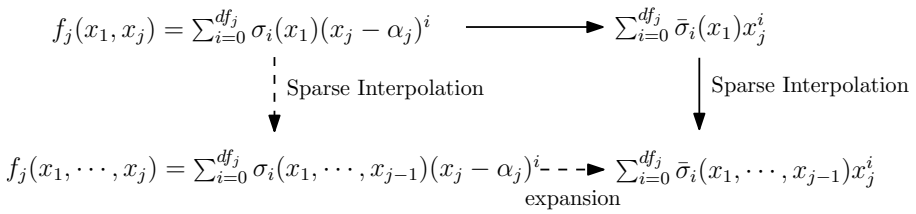

Fig. 1. Dashed arrows: Algorithm 3 [11], expression swell occurs at the expansion step. Lined arrows: CMSHL (Algorithm 4).

lifting step:

$$f_j(x_1, \cdots, x_j) = \sum_{i=0}^{df_j} \sigma_i(x_1, \cdots, x_{j-1})(x_j - \alpha_j)^i = \sum_{i=0}^{df_j} \bar{\sigma}_i(x_1, \cdots, x_{j-1})x_j^i, \quad (1)$$

where $df_j = \deg(f_j, x_j)$. There are two routes to recover $\bar{\sigma}_i(x_1, \cdots, x_{j-1})$ in (1) from its bivariate image $f_j(x_1, x_j)$. One route is to first recover $\sigma_i(x_1, \cdots, x_{j-1})$ from $\sigma_i(x_1)$ using sparse interpolation and then expand to get $\bar{\sigma}_i(x_1, \cdots, x_{j-1})$ in (1) (through the dashed arrows in Fig. 1). This has been done previously in [11]. In our new algorithm (CMSHL), bivariate images are expanded first and then the coefficients $\bar{\sigma}_i(x_1, \cdots, x_{j-1})$ are recovered directly from $\bar{\sigma}_i(x_1)$ to get the final expanded form. This is through the lined arrows in Fig. 1. Multivariate polynomial expansions are avoided where expression swells can occur.

Our solution is presented in Algorithm 4 (CMSHL). CMSHL also has a significant advantage for parallelization, however, it only uses the weak SHL assumption (defined in Section 3.2) during a sparse interpolation. It can not use the strong SHL assumption (defined in Section 3.1) as in MTSHL to reduce the number of terms in a loop for a typical average case.

3 Complexity Analyses

For both MTSHL and CMSHL, the number of arithmetic operations in $\mathbb{Z}_p$ are bounded for the worst-case, along with the failure probabilities. We first need the Schwartz-Zippel Lemma [20, 17]:

Lemma 1. *Let F be a field and $f \neq 0$ be a polynomial in $F[x_1, x_2, \cdots, x_n]$ with total degree d and let $S \subseteq F$. Then the number of roots of f in S^n is at most $d|S|^{n-1}$. Hence if β is chosen at random from S^n then $\Pr[f(\beta) = 0] \leq \frac{d}{|S|}$.*

3.1 MTSHL

MTSHL uses the strong SHL assumption to solve the multivariate Diophantine equations (MDPs) in a loop. The following lemma was proved in [9]:

Algorithm 4 CMSHL: Hensel lifting x_j via bivariate Hensel lifting.

Input: A prime p, $\alpha_j \in \mathbb{Z}_p$, $a_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ monic in x_1, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ s.t. $a_j(x_1, ..., x_{j-1}, \alpha_j) = f_{j-1}g_{j-1}$ with $j > 2$.
Output: $f_j, g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ s.t. $a_j = f_j g_j$ where $f_j(x_j = \alpha_j) = f_{j-1}$ and $g_j(x_j = \alpha_j) = g_{j-1}$; Otherwise, **return** FAIL.
Let $f_{j-1} = x_1^{df} + \sum_{i=0}^{df-1} \sigma_i(x_2, ..., x_{j-1})x_1^i$, $\sigma_i = \sum_{k=1}^{s_i} c_{ik}M_{ik}$ and $g_{j-1} = x_1^{dg} + \sum_{i=0}^{dg-1} \tau_i(x_2, ..., x_{j-1})x_1^i$, $\tau_i = \sum_{k=1}^{t_i} d_{ik}N_{ik}$, where M_{ik},N_{ik} are the monomials in σ_i, τ_i respecitvely.
Pick $(\beta_2, \cdots, \beta_{j-1}) \in \mathbb{Z}_p^{j-2}$ at random.
Compute monomial evaluation sets ..... $\mathcal{O}((j-2)(\#f + \#g + d_{\max}))$
$\mathcal{S} = \{S_i = \{m_{ik} = M_{ik}(\beta_2, \cdots, \beta_{j-1}), 1 \le k \le s_i\}, 0 \le i \le df - 1\}$ and
$\mathcal{T} = \{T_i = \{n_{ik} = N_{ik}(\beta_2, \cdots, \beta_{j-1}), 1 \le k \le t_i\}, 0 \le i \le dg - 1\}$.
if any $|S_i| \neq s_i$ **or** any $|T_i| \neq t_i$ **then return** FAIL(1) **end if**
Let s be the maximum of s_i and t_i.
for k from 1 to s **in parallel do**
    Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
    $A_k, F_k, G_k \leftarrow a_j(x_1, Y_k, x_j), f_{j-1}(x_1, Y_k), g_{j-1}(x_1, Y_k)$. .. $\mathcal{O}(s(\#f + \#g + \#a))$
    if $\gcd(F_k, G_k) \neq 1$ **then return** FAIL(2) **end if** // unlucky evalution
    $f_k, g_k \leftarrow$ *BivariateHenselLift*$(A_k, F_k, G_k, \alpha_j, p)$. ..... $\mathcal{O}(s(d_1^2 d_j + d_1 d_j^2))$
end for
Let $f_k = x_1^{df} + \sum_{l=1}^{\mu} \alpha_{kl}\tilde{M}_l(x_1, x_j)$ for $1 \le k \le s$, where $\mu \le d_1 d_j$.
for l from 1 to μ **in parallel do**
    $i \leftarrow \deg(\tilde{M}_l, x_1)$.
    Solve the $s_i \times s_i$ linear system for c_{lk} ..... $\mathcal{O}(sd_j\#f)$

$$\left\{ \sum_{k=1}^{s_i} m_{ik}^n c_{lk} = \alpha_{nl} \text{ for } 1 \le n \le s_i \right\}$$

end for
Construct $f_j \leftarrow x_1^{df} + \sum_{l=1}^{\mu} \left(\sum_{k=1}^{s_i} c_{lk} M_{ik}(x_2, ..., x_{j-1}) \right) \tilde{M}_l(x_1, x_j)$.
Similarly, construct g_j. ..... $\mathcal{O}(sd_j\#g)$
if $a_j = f_j g_j$ **then return** (f_j, g_j) **else return** FAIL(3) **end if**

Lemma 2. *Let $f \in \mathbb{Z}_p[x_1, \cdots, x_n]$ and let α be a randomly chosen element in $\mathbb{Z}_p$. Let $f = \sum_{i=0}^{d_n} \sigma_i(x_1, \cdots, x_{n-1})(x_n - \alpha)^i$ where $d_n = \deg(f, x_n)$. Then,*

$$\Pr[\mathrm{Supp}(\sigma_{i+1}) \not\subseteq \mathrm{Supp}(\sigma_i)] \le |\mathrm{Supp}(\sigma_{i+1})| \frac{d_n - i}{p - d_n + i + 1} \text{ for } 0 \le i < d_n.$$

The assumption that $\mathrm{Supp}(\sigma_i) \subseteq \mathrm{Supp}(\sigma_{i-1})$ for $1 \le i \le d_n$ is called the **strong SHL assumption** in [9, 12]. In Section 3.2 our new algorithm will assume $\mathrm{Supp}(\sigma_i) \subseteq \mathrm{Supp}(\sigma_0)$ for $1 \le i \le d_n$. This assumption is called the **weak SHL assumption** in [9, 12].

Step 11 of Algorithm 1 applies the strong SHL assumption by using $\mathrm{Supp}(\sigma_f) = \mathrm{Supp}(\sigma_{i-1})$ and $\mathrm{Supp}(\tau_f) = \mathrm{Supp}(\tau_{i-1})$ as the supports for σ_i and τ_i. Therefore we only solve systems of linear equations for the coefficients. This is the key feature to solve the MDPs via Algorithm 2, which we shall analyze in the following.

3.1.1 The failure probability of the MDPs

There are two places where Algorithm 2 can return FAIL intermediately: line 7 and line 11. The failuare probabilities are bounded as follows. Proofs follow [12].

Proposition 1. *Let p be a large prime, $d = \deg(a)$ and s be the number defined in line 4 of Algorithm 2. When Algorithm 1 calls Algorithm 2 with inputs $(u, w, c, \sigma_f, \tau_f) = (g_{j-1}, f_{j-1}, c_i, \sigma_{i-1}, \tau_{i-1})$, if $\mathrm{Supp}(\sigma_i) \subseteq \mathrm{Supp}(\sigma_{i-1})$ and $\mathrm{Supp}(\tau_i) \subseteq \mathrm{Supp}(\tau_{i-1})$, for $i = 1, 2, 3, \cdots$, then Algorithm 2 fails to compute (σ_i, τ_i) for the MDP $\sigma_i g_{j-1} + \tau_i f_{j-1} = c_i$ with a probability less than*

$$\underbrace{\frac{d\, s(\#f_{j-1} + \#g_{j-1})}{2(p-1)}}_{\text{line 7}} + \underbrace{\frac{d^2 s^2}{p-1}}_{\text{line 11}}. \tag{2}$$

Proof. For line 7, let $\Delta_i = \prod_{1 \le l < k \le s_i} (M_{il} - M_{ik})$, where M_{il}, M_{ik} are monomials in S defined in line 6. Let $\Delta = \prod_{i=0}^{d\sigma} \Delta_i$. Then $\Delta(\beta_2, \ldots, \beta_{j-1}) = 0$ implies $\Delta_i(\beta_2, \ldots, \beta_{j-1}) = 0$ for some i so that not all monomial evaluations are distinct. Also, $\deg(M_{il}) < d$ for each monomial in $\mathcal{S}$. Thus,

$$\deg(\Delta) < \sum_{i=0}^{d_\sigma} d \binom{s_i}{2} \le \frac{d\, s}{2} \sum_{i=0}^{d_\sigma} (s_i - 1) < \frac{d\, s \# f_{j-1}}{2}.$$

By Lemma 1,

$$\Pr[\Delta(\beta_2, \cdots, \beta_{j-1}) = 0] \le \frac{\deg(\Delta)}{p-1} < \frac{d\, s \# f_{j-1}}{2(p-1)}.$$

Similarly, the monomial evaluations for τ are considered.
To solve the Diophantine equation in line 12, we need

$$\gcd(u(x_1, Y_k), w(x_1, Y_k)) = \gcd(g_{j-1}(x_1, Y_k), f_{j-1}(x_1, Y_k)) = 1.$$

Let $R = \mathrm{res}(g_{j-1}, f_{j-1}, x_1) \in \mathbb{Z}_p[x_2, \cdots, x_{j-1}]$. Since f_{j-1} and g_{j-1} are monic in x_1, the univariate Diophantine solver returns FAIL if

$$\gcd(g_{j-1}(x_1, Y_k), f_{j-1}(x_1, Y_k)) \neq 1 \iff R(Y_k) = 0.$$

Let $S = \prod_{k=1}^{s} R(x_2^k, x_3^k, \cdots, x_{j-1}^k)$. Since $\deg(f_{j-1}) < d$ and $\deg(g_{j-1}) < d$, $\deg(R) < d^2$ and

$$\deg(S) = \sum_{k=1}^{s} k \deg(R) < \sum_{k=1}^{s} k d^2 = \frac{d^2 s(s+1)}{2}.$$

By Lemma 1,

$$\Pr[R(Y_k) = 0 \text{ for some } k] = \Pr[S(\beta_2, \cdots, \beta_{j-1}) = 0] \le \frac{\deg(S)}{p-1} < \frac{d^2 s^2}{p-1}.$$

Adding the failure probabilities at line 7 and 11, we obtain the result. □

At the end of Algorithm 2, $\sigma u + \tau w = c$ can be checked probabilistically with a single evaluation point. If Algorithm 2 returns FAIL at line 19, the support in either σ or τ was wrong (strong SHL assumption fails). By Lemma 2, Algorithm 1 fails at the j^{th} Hensel lifting step due to a wrong support in σ_i with a probability no more than

$$\sum_{i=0}^{d_j-1} |\text{supp}(\sigma_{i+1})| \frac{d_j - i}{p - d_j + i + 1} \leq \#f_{j-1} \sum_{i=0}^{d_j-1} \frac{d_j - i}{p - d_j + i + 1} < \frac{d_j(d_j+1)\#f_{j-1}}{2(p - d_j + 1)},$$

where $d_j = \deg(a_j, x_j)$.

Note that the number s in Proposition 1 varies since MDPs are called in a loop from Algorithm 1. We denote $s_{j,i}$ as the maximum number of monomials in the coefficients of σ_{i-1} and τ_{i-1} in x_1 for the i^{th} call of the MDP in the j^{th} Hensel lifting step. Let $s_j = \max_i(s_{j,i})$ and $T_{fg_{j-1}} = \max(\#f_{j-1}, \#g_{j-1})$. We have $d_j \leq d$. Adding up the failure probabilities at line 7, 11 and 19, we obtain the failure probability at the j^{th} Hensel lifting step:

Proposition 2. *Let p be a large prime, $d = \deg(a)$, $s_j = \max_i(s_{j,i})$ and $T_{fg_{j-1}} = \max(\#f_{j-1}, \#g_{j-1})$. Algorithm 1 (MTSHL) fails to compute f_j, g_j from f_{j-1}, g_{j-1} at the j^{th} Hensel lifting step ($j > 2$) via Algorithm 2 with a probability less than*

$$\frac{d^2 s_j (T_{fg_{j-1}} + d\, s_j) + d^2 T_{fg_{j-1}} + d T_{fg_{j-1}}}{p - d + 1}. \tag{3}$$

For the whole MTSHL process (for $2 \leq j \leq n$), we have $\#f_{j-1} \leq \#f$, $\#g_{j-1} \leq \#g$.

Proposition 3. *Let p be a large prime, n be the number of variables in a, $d = \deg(a)$, $s_{\max} = \max(s_j)$ and $T_{fg} = \max(\#f, \#g)$. MTSHL (the j^{th} Hensel lifting step as in Algorithm 1) fails to solve the MDP via sparse interpolation (Algorithm 2) with a probability less than*

$$\frac{(n-2)\left(d^2 s_{\max}(T_{fg} + d\, s_{\max}) + d^2 T_{fg} + d T_{fg}\right)}{p - d + 1}. \tag{4}$$

We illustrate the probability in Proposition 3 for a typical large factorization problem. Let $n = 10$, $d = 10^2$, $T_{fg} = 10^4$ and $s_{\max} = 10^2$. If p is a 64-bit prime $\approx 1.8 \times 10^{19}$, then MTSHL fails with probability less than 8.72×10^{-9}. Thus for p sufficiently large, the failure probability is low.

3.1.2 The complexity of the MDP

After discussing the failure probabilities, it remains to bound the number of arithmetic operations in $\mathbb{Z}_p$. We have the complexity of the MDP as follows:

Theorem 1. *Let p be a large prime, s be the number defined in line 4 of Algorithm 2, $d_1 = \deg(a, x_1)$ and a_j $(j > 2)$ be monic in x_1. When Algorithm 1 calls Algorithm 2, if the strong SHL assumption holds, then with a failure probability less than in (2), the number of arithmetic operations in $\mathbb{Z}_p$ for solving the MDP $\sigma_i g_{j-1} + \tau_i f_{j-1} = c_i$ for $i = 1, 2, \cdots$ in the worst case is*

$$\mathcal{O}(s(\#a_j + d_1^2)). \tag{5}$$

Proof. Appendix A.

3.1.3 The complexity of MTSHL

Now we return to the analysis of Algorithm 1 – Hensel lifting x_j with multivariate Diophantine equations. One bottleneck of Algorithm 1 is the error computation step at line 14. There is an expression swell of f_j and g_j at line 13 of up to a factor of $d_j = \deg(a, x_j)$. We have the complexity at the j^{th} Hensel lifting step:

Theorem 2. *Let p be a large prime, $d_1 = \deg(a, x_1)$, $d_j = \deg(a, x_j)$ and $s_j = \max_i s_{j,i}$. With a failure probability less than in (3), the number of arithmetic operations in $\mathbb{Z}_p$ for the j^{th} Hensel lifting step (via Algorithm 1) in the worst case is*

$$\mathcal{O}(\underbrace{d_j^2 \#a_j}_{\text{line 7,13}} + \underbrace{d_j s_j (\#a_j + d_1^2)}_{\text{MDP}} + \underbrace{d_j^3 \#f_{j-1} \#g_{j-1}}_{\text{error comp.}}). \tag{6}$$

Proof. To compute coeff(error, $(x_j - \alpha_j)^i$) in step 7, using repeated differentiation and evaluation costs $O(i\#\text{error})$. The total cost is $\mathcal{O}(d_j^2 \#a_j)$.

The total cost of sparse interpolation in step 11 is $\mathcal{O}(d_j s_j (\#a_j + d_1^2))$, from Theorem 1.

The total cost of adding the factors in step 13 is $\mathcal{O}\left(\sum_{i=1}^{d_j} i(\#f_{j-1} + \#g_{j-1})\right)$, which is $\mathcal{O}(d_j^2(\#f_{j-1} + \#g_{j-1}))$.

The total cost of error computation in step 14 is $\mathcal{O}\left(\sum_{k=1}^{d_j} \#f_j^{(k)} \#g_j^{(k)}\right) = \mathcal{O}\left(\sum_{i=1}^{d_j} (i\#f_{j-1})(i\#g_{j-1})\right) \subseteq \mathcal{O}(d_j^3 \#f_{j-1} \#g_{j-1})$.

We assume $\#f_{j-1} \leq \#a_j$, $\#g_{j-1} \leq \#a_j$, the total cost for Algorithm 1 is

$$\mathcal{O}(\underbrace{d_j^2 \#a_j}_{\text{line 7,13}} + \underbrace{d_j s_j (\#a_j + d_1^2)}_{\text{MDP}} + \underbrace{d_j^3 \#f_{j-1} \#g_{j-1}}_{\text{error comp.}}). \ \square$$

In Theorem 2, the expression swell appears as the factor of d_j^2. On average the expression swell is much less. For sparse factors, we know that $\#f_{j-1} \lessapprox \#f_j$ for $n/2 \leq j \leq n$ [12]. The complexity of the whole MTSHL process is given in the following:

Theorem 3. *Let p be a large prime, $a \in \mathbb{Z}_p[x_1, \cdots, x_n]$ monic in x_1, $\alpha = (\alpha_2, \cdots, \alpha_n) \in {\mathbb{Z}_p}^{n-1}$ be a random evaluation point. Assume $\gcd(f_1, g_1) = 1$. Then with a failure probability less than in (4), the total number of arithmetic*

operations in $\mathbb{Z}_p$ for lifting f_1,g_1 to f_n,g_n in $n-1$ steps using MTSHL (Algorithm 1) in the worst case is

$$\mathcal{O}(\underbrace{d_1^2 d_2 + d_1 d_2^2}_{\text{first BHL}} + (n-2)(\underbrace{d_{\max}^2 \#a + s_{\max} d_{\max}(\#a + d_1^2) + d_{\max}^3 \#f \#g}_{\text{MTSHL}})). \quad (7)$$

where $d_i = \deg(a, x_i)$ for $1 \le i \le n$, $d_{\max} = \max_{i=3}^{n}(d_i)$ and $s_{\max} = \max(s_j)$.

3.2 CMSHL

In Algorithm CMSHL, the weak SHL assumption is used instead of the strong SHL assumption. Similar to Lemma 2, we have the following (proof follows [9]):

Lemma 3. *Let $f \in \mathbb{Z}_p[x_1, \cdots, x_n]$ and let α be a randomly chosen element in $\mathbb{Z}_p$. Let $f = \sum_{i=0}^{d_n} \sigma_i(x_1, \cdots, x_{n-1})(x_n - \alpha)^i$ where $d_n = \deg(f, x_n)$. Then,*

$$\Pr[\mathrm{Supp}(\sigma_i) \not\subseteq \mathrm{Supp}(\sigma_0)] \le |\mathrm{Supp}(\sigma_i)| \frac{d_n}{p - d_n + i} \text{ for } 1 \le i \le d_n.$$

The assumption that $\mathrm{Supp}(\sigma_i) \subseteq \mathrm{Supp}(\sigma_0)$ for $1 \le i \le d_n$ is called the **weak SHL assumption**.

3.2.1 The failure probability of CMSHL

For the j^{th} Hensel lifting step, by Lemma 3, the failure probability due to a wrong support in either f_j or g_j (Algorithm 4 fails at line 21) is bounded by

$$(\#f_{j-1} + \#g_{j-1}) \sum_{i=1}^{d_j} \frac{d_j}{p - d_j + i} \le \frac{d_j^2(\#f_{j-1} + \#g_{j-1})}{p - d_j + 1}.$$

The number s defined in line 7 of Algorithm 4 is equivalent to $s_j = \max(s_{j,i})$ in MTSHL. We denote s_j as the number s in line 7 of Algorithm 4 at the j^{th} Hensel lifting step. Identical to MTSHL (Proposition 2), the failure probabilities at line 6 and 11 are

$$\underbrace{\frac{d\, s_j(\#f_{j-1} + \#g_{j-1})}{2(p-1)}}_{\text{line 6}} + \underbrace{\frac{d^2 s_j^2}{p-1}}_{\text{line 11}}.$$

Adding the failure probabilities at line 6, 11 and 21, we have the failure probability at the j^{th} Hensel lifting step for Algorithm 4 (CMSHL):

Proposition 4. *Let p be a large prime, $d = \deg(a)$, $T_{fg_{j-1}} = \max(\#f_{j-1}, \#g_{j-1})$ and s_j be the number s defined in line 7 of Algorithm 4 at the j^{th} Hensel lifting step. Then Algorithm 4 fails to compute f_j,g_j from f_{j-1},g_{j-1} at the j^{th} Hensel lifting step ($j > 2$) with a probability less than*

$$\frac{d\, s_j(T_{fg_{j-1}} + d\, s_j) + 2d^2 T_{fg_{j-1}}}{p - d + 1}. \quad (8)$$

3.2.2 The complexity of CMSHL

Theorem 4. *Let p be a large prime, $d_1 = \deg(a, x_1)$, $d_j = \deg(a, x_j)$ and s_j be the number s defined in line 7 of Algorithm 4 for the j^{th} Hensel lifting step. With a failure probability less than in (8), the number of arithmetic operations in $\mathbb{Z}_p$ for the j^{th} Hensel lifting step (via Algorithm 4) in the worst case is*

$$\mathcal{O}(d_j s_j(\#f_{j-1} + \#g_{j-1} + d_1^2 + d_1 d_j) + s_j \#a_j). \tag{9}$$

Proof. Appendix B.

For the whole process of CMSHL (for $2 \le j \le n$), we have the following:

Theorem 5. *Let p be a large prime, $a \in \mathbb{Z}_p[x_1, \cdots, x_n]$ monic in x_1, $\alpha = (\alpha_2, \cdots, \alpha_n)$ be a randomly chosen evaluation point from $\mathbb{Z}_p^{n-1}$, f, g be the monic irreducible factors of a, $f_1 = f(x_1, \alpha)$, $g_1 = g(x_1, \alpha)$ be the image polynomials with $\gcd(f_1, g_1) = 1$. With a failure probability less than*

$$\frac{(n-2)(d\, s_{\max}(T_{fg} + d\, s_{\max}) + 2d^2 T_{fg})}{p - d + 1}, \tag{10}$$

the number of arithmetic operations in $\mathbb{Z}_p$ for lifting f_1,g_1 to f_n,g_n in $n-1$ steps using CMSHL (Algorithm 4) in the worst case is

$$\mathcal{O}(\underbrace{d_1^2 d_2 + d_1 d_2^2}_{\text{first BHL}} + (n-2)\underbrace{(s_{\max} d_{\max}(\#f + \#g + d_1^2 + d_1 d_{\max}) + s_{\max}\#a)}_{\text{CMSHL}}), \tag{11}$$

where $d = \deg(a)$, $d_i = \deg(a, x_i)$ for $1 \le i \le n$, $d_{\max} = \max_{i=3}^{n}(d_i)$, $s_{\max} = \max(s_j)$ and $T_{fg} = \max(\#f, \#g)$.

4 Experimental Results

We have implemented our factorization algorithm in the C programming language and parallelized parts of it for multi-core computers using Cilk C [3]. Cilk uses the fork-join idiom for parallel programming. Our C code and Cilk C code is freely available on the web at `http://www.cecm.sfu.ca/CAG/code/CASC2020`

Following the recommendation in [12] we interpolate $f(x_1, \ldots, x_j)$ (using sparse interpolation) from trivariate images $f(x_1, x_2, \beta^i, x_j)$ instead of from bivariate images $f(x_1, \beta^i, x_j)$. To obtain a trivariate image we interpolate x_2 using dense interpolation from bivariate images $f(x_1, \gamma_k, \beta^i, x_j)$ which we obtain using bivariate Hensel lifting. Although this increases the cost of computing images by a factor of $\deg(f, x_2)$, using trivariate images typically reduces s_j in equation (9) which speeds up all other parts of our algorithm. We refer the reader to [12] for an analysis of the expected reduction in s_j.

We give three sets of timings for our factorization code. The first set (see Table 3) is for factors with a low degree of 7 and an increasing number of terms t ($\#f = \#g = t$). For this case evaluating $a(x_1, x_2, \beta^i, x_j)$ is the bottleneck of our algorithm. The second set (see Table 4) is for factors with a fixed number

					New times (1 core)			New times (16 cores)			Maple	Maple	Magma
n	d	t	s	$f \times g$	total	hensel	eval	total	hensel	eval	2019	2017	V2.25-5
6	7	500	17	0.025	0.084	0.012	0.029	0.081	0.016	0.007	1.897	33.77	43.21
6	7	1000	30	0.107	0.340	0.021	0.170	0.169	0.028	0.027	4.540	95.48	50.38
6	7	2000	47	0.451	1.199	0.033	0.768	0.321	0.044	0.114	97.21	186.7	195.6
6	7	4000	81	1.932	3.583	0.055	2.632	0.543	0.065	0.281	139.4	325.4	777.0
6	7	8000	144	8.249	8.778	0.101	7.107	1.248	0.125	0.830	201.1	470.5	1958.0
9	7	500	14	0.025	0.119	0.013	0.031	0.094	0.013	0.005	4.699	2794.2	849.2
9	7	1000	28	0.108	0.493	0.021	0.204	0.232	0.066	0.031	15.57	16094	915.8
9	7	2000	50	0.449	2.169	0.034	1.313	0.433	0.042	0.160	3597.7	>32GB	9082.8
9	7	4000	99	1.963	13.94	0.067	10.47	1.570	0.076	0.816	>32GB	NA	15444
9	7	8000	178	8.244	88.39	0.121	74.14	8.313	0.138	5.575	NA	NA	>32GB

Table 3. Real timings in CPU seconds for low degree d and increasing terms t.

of terms and an increasing degree d. For these problems Hensel lifting becomes the bottleneck. To address this we use Monagan's $O(d^3)$ method [14] for Hensel lifting in $\mathbb{Z}_p[x, y]$. The third set (see Table 5) is for polynomials where the factor f has a lot more terms than g. For these problems evaluation and solving Vandermonde systems are the bottlenecks. To solve the Vandermonde systems we use Zippel's linear space quadratic time method in [22].

All experiments were performed on a server with two Intel E5-2660 8 core CPUs running at 2.2GHz (base) and 3.0GHz (turbo) hence the maximum theoretical parallel speedup is a factor of $16 \times 2.2/3.0 = 11.7$.

In Tables 3, 4 and 5 the factors f and g are of the form $x_1^d + \sum_{i=2}^{t-1} a_i \prod_{j=1}^{n} x_j^{e_{ji}}$ with coefficients a_i chosen randomly from $[1, 999]$ and exponents e_{ji} chosen randomly from $[0, d-1]$. The time in column $f \times g$ is the time our C code takes to multiply $a = f \times g$ using an algorithm with arithmetic complexity $O(\#f\,\#g)$.

Because the factors are monic and have many terms, almost all of the factorization time is in multivariate Hensel lifting. The timings for our algorithm are for Hensel lifting x_n the last variable only, which is most of the time. The quantity s in column 4 is the number of images needed to interpolate $x_3, \ldots, x_n$.

For Maple we report timings for Maple 2017 and Maple 2019. Maple 2017 and Magma 2.25-5 are both using Wang's organization of MHL as described in Chapter 6 of [5]. Maple 2019 is using Monagan and Tuncer's sparse Hensel lifting from [9, 12]. These algorithms do many computations with multivariate polynomials in $\mathbb{Z}_p[x_1, \ldots, x_j]$ including many multiplications and divisions. In contrast, our algorithm does no arithmetic with multivariate polynomials.

In Tables 3, 4 and 5 we report the total time of our new algorithm in column total, the time evaluating $a(x_1, x_2, \beta^i, x_n)$ for $1 \le i \le s$ in column eval, the time in bivariate Hensel lifting in column hensel, and for Table 5, the time solving the Vandermonde systems in column solve. Timings are given for our Cilk C code for 1 core and 16 cores.

					New times (1 core)			New times (16 cores)			Maple	Maple	Magma
n	d	t	s	$f \times g$	total	hensel	eval	total	hensel	eval	2019	2017	v2.25-5
6	10	500	10	0.026	0.079	0.022	0.017	0.069	0.020	0.004	3.068	466.8	134.7
6	15	500	6	0.025	0.101	0.051	0.011	0.094	0.036	0.004	8.206	11002	610.1
6	20	500	5	0.025	0.168	0.117	0.012	0.101	0.036	0.004	18.77	51325	27317.
6	40	500	3	0.025	0.669	0.617	0.011	0.272	0.205	0.008	148.7	NA	29.04
6	60	500	3	0.025	2.083	2.025	0.014	0.583	0.519	0.010	545.2	NA	371.4
6	80	500	3	0.025	5.644	5.586	0.014	0.950	0.892	0.010	1210.9	NA	1242.1
6	100	500	2	0.025	7.740	7.687	0.008	1.375	1.303	0.008	NA	NA	NA
6	10	2000	30	0.455	1.434	0.070	0.737	0.258	0.043	0.056	675.11	1889.3	908.0
6	15	2000	18	0.455	1.327	0.168	0.488	0.341	0.100	0.060	3905.7	63082	9317.1
6	20	2000	12	0.455	1.336	0.329	0.332	0.335	0.136	0.042	4677.2	$> 10^5$	17339
6	40	2000	6	0.455	2.853	1.999	0.183	0.686	0.472	0.038	>32GB	NA	$> 10^5$
6	60	2000	6	0.455	8.940	8.071	0.203	1.313	1.106	0.052	NA	NA	NA
6	80	2000	4	0.455	15.17	14.34	0.158	2.565	2.279	0.084	NA	NA	NA
6	100	2000	3	0.455	21.77	20.92	0.173	2.644	2.357	0.086	NA	NA	NA

Table 4. Real timings in CPU seconds for increasing degree d and fixed t.

n	d	t	$\#g$	s	$f \times g$	total	hensel	eval	solve	total	hensel	eval	solve
9	7	10000	20	212	0.043	0.871	0.156	0.340	0.287	0.350	0.155	0.060	0.039
9	7	20000	20	409	0.076	2.641	0.254	1.107	1.108	0.663	0.256	0.122	0.096
9	7	40000	20	789	0.135	9.465	0.475	4.243	4.175	1.917	0.477	0.480	0.361
9	7	80000	20	1503	0.258	34.16	0.920	15.68	16.33	4.782	0.913	1.362	1.373
9	7	160000	20	2984	0.499	132.3	1.791	62.13	64.37	13.67	1.844	5.586	5.244

Table 5. Real timings in CPU seconds for increasing $\#f = t$ and $\#g = 20$.

Tables 3 and 5 show good parallel speedup for the evaluations $a(x_1, x_2, \beta^i, x_n)$. Table 4 shows that for higher degree polynomials the Hensel lifting dominates. To obtain the parallel speedups for the Hensel lifting in Table 4 we parallelize the evaluations of $a(x_1, x_2, \beta^i, x_n)$ at $x_2 = \gamma_k$ for different k as well as the bivariate Hensel Lifts in $\mathbb{Z}_p[x_1, x_n]$.

Table 5 shows that when one factor is much larger than the other, the time solving Vandermonde systems becomes significant. The solving time is not reported in Tables 3 and 4 because it is insignificant.

The timings in Tables 3, 4, and 5 agree with our analysis for CMSHL in Theorem 4. In Table 3, for example, when $n = 9$ and t increases from 2000 to 4000, $\#a_j$ is quadrupled and s_j is doubled, we see the evaluation time for 1 core increases by a factor of $10.47/1.313 = 7.97 \approx 8$. This agrees with the term $s_j \#a_j$ in (9). In Table 4, when $t = 2000$ and d increases from 60 to 100, we expect the time for Hensel lifting at $d = 100$ to be $\frac{1}{2}(100/60)^3 \cdot 8.071 = 18.68$ which is close to the result 20.92. In Table 5, when t and s are doubled, both timings for evaluations and solving Vandermonde systems are quadrupled as expected.

5 Implementation Notes

To store the multivariate polynomial $a = \sum_{i=1}^{t} a_i M_i(x_1, \ldots, x_n)$ we encode the monomials M_i in 64 bit integers m_i. We store a as the triple (A, X, t) where

$$A = \boxed{a_1}\boxed{a_2}\boxed{\ldots}\boxed{a_t} \quad \text{and} \quad X = \boxed{m_1}\boxed{m_2}\boxed{\ldots}\boxed{m_t}$$

are stored as arrays. Although monomial packing limits the degree and number of variables that our software can handle it improves code performance.

One of the advantages of our algorithm is that there are no multivariate polynomial multiplications and divisions. The most time consuming operation is evaluation which is linear in the number of terms. We compute $a(x_1, x_2, \beta^i, x_n)$ for $\beta \in \mathbb{Z}_p^{n-3}$ for $1 \le i \le s$. We have parallelized these evaluations. We parallelize each evaluation $a(x_1, x_2, \beta^i, x_n)$ in blocks and do two evaluations at a time.

We also execute the bivariate Hensel lifts in parallel and we solve the Vandermonde linear systems in parallel. To avoid memory bottlenecks, we use in-place algorithms for all parallel tasks. A routine is in-place if it, and all the subroutines it calls, allocate no memory. They work in the memory of the input and output. This means that our Cilk tasks are not simultaneously trying to allocate and de-allocate memory. We give an example of an in-place algorithm.

The following conversion occurs at the end of bivariate Hensel lifting. We have polynomials $a_0(x), a_1(x), \ldots, a_d(x)$ in $\mathbb{Z}_p[x]$, a non-zero element $\alpha \in \mathbb{Z}_p$, and we want to expand the bivariate polynomial

$$f(x, y) = \sum_{i=0}^{d} a_i(x)(y - \alpha)^i$$

that is, we want to compute new polynomials $\bar{a}_i \in \mathbb{Z}_p[x]$ such that $f(x, y) = \sum_{i=0}^{d} \bar{a}_i(x) y^i$ in $\mathbb{Z}_p[x, y]$. One way to expand $f(x, y)$ is to use Horner's rule

$$f(x, y) = a_0(x) + (y - \alpha)\left[a_1(x) + (y - \alpha)\left[a_2(x) + \cdots + (y - \alpha)a_d(x)\ldots\right]\right].$$

Coding this in Maple or Magma will cause $2d$ pieces of memory to be allocated for the intermediate products and sums. To code this in C we have to handle the memory explicitly. How we do this depends the data structure we use for storing polynomials in $\mathbb{Z}_p[x, y]$.

Let $f \in \mathbb{Z}_p[x, y]$, $dx = \deg(f, x)$, $dy = \deg(f, y)$ and $d_i = \deg(f, x_i)$. We store f as a pair (D, A) where D is an array of integers storing the degree information $[d_y, d_0, d_1, \ldots, d_{dy}]$ and A is an array of arrays storing $[a_0(x), a_1(x), \ldots, a_d(x)]$. To do the conversion we use this version of Horner's rule

for $i = d - 1, d - 2, \ldots, 0$ do
  for $j = i, i + 1, \ldots, d - 1$ do
    $a_j(x) := a_j(x) - \alpha\ a_{j+1}(x)$.

To implement this in-place we use the inplace routine `polsubmul`$(a, da, b, db, \alpha, p)$ from our $\mathbb{Z}_p[x]$ library which computes $a(x) := a(x) - \alpha\, b(x)$ in the memory of a and returns the degree of the result. `polsubmul` assumes the size of the array a is big enough to hold b. Assuming each array A_i has space for $1 + dx$ coefficients in $\mathbb{Z}_p$, we can do the conversion in the memory of (D, A) with

```
for( i=D[0]-1; i>=0; i-- )
   for( j=i; j<d; j++ )
      D[j+1] = polsubmul(A[j],D[j],A[j+1],D[j+1],alpha,p);
```

We have coded every subroutine in our bivariate Hensel lift to run in-place so that our bivariate Hensel lift can also be made in-place. In this way, when we run bivariate Hensel lifts in parallel, they all run in their own pre-allocated memory.

6 Conclusion

Algorithm 1 is the basis for several polynomial factorization algorithms, including Wang's method from [18] which is used in most computer algebra systems today, and Monagan and Tuncer's method [9, 12] which is now used in Maple. In this work we observed an expression swell in Algorithm 1 that is linear in the worst case. We presented a new sparse Hensel lifting algorithm CMSHL that avoids the expression swell. CMSHL, which is based on the method in [11], is suited to parallelization because it reduces multivariate polynomial factorization to many polynomial evaluations, many bivariate Hensel lifts, and solving many Vandermonde systems.

Our Cilk C implementation of CMSHL shows good parallel speedup for these three steps. The code is also much faster than the Maple and Magma factorization algorithms for the large factorization problems we tested, mainly because it does not do any multivariate polynomial arithmetic. We have also given a worst case complexity analysis for CMSHL and have determined its failure probability. For factors with many terms and not too high degree, as in Table 3, our experiments show that evaluation is the bottleneck. This agrees with the term $s_j \# a_j$ in equation (9) in our complexity analysis. Thus further improvement will need to consider these evaluations.

For future work, we would like to use CMSHL to factor polynomials represented by black boxes in the spirit of Kaltofen and Trager [7] and Diaz and Kaltofen [2].

Appendix A Proof of Theorem 1

We bound the total number of arithmetic operations in $\mathbb{Z}_p$ for the worst case in Algorithm 2. Let s be the number defined in line 4, $d_{\max} = \max_{i=2}^{j-1} \deg(a, x_i)$ and $df_{\max} = \max_{i=2}^{j-1} \deg(f, x_i)$.

For step 6, one way to evaluate the monomials is to create a table of powers for each variable $x_2, \cdots, x_{j-1}$, as shown in Fig. 2. It takes $\sum_{i=2}^{j-1}(d\sigma_{f_i} - 1) \leq (j-2)(df_{\max}-1)$ multiplications to compute the table, where $d\sigma_{f_i} = \deg(\sigma_f, x_i)$. After creating the table, it takes $\mathcal{O}\left((j-3)\sum_{i=0}^{d\sigma} s_i\right) = \mathcal{O}((j-3)\#\sigma_f)$ multiplications to evaluate monomials in $\mathcal{S}$. Similarly for the evaluations in $\mathcal{T}$. Thus, the total cost is $\mathcal{O}((j-2)(\#\sigma_f + \#\tau_f + d_{\max}))$.

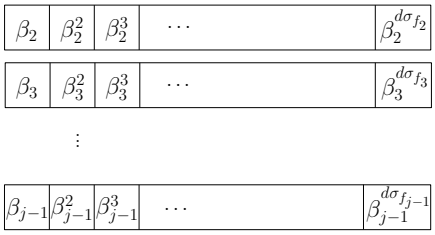

Fig. 2. Evaluation table for variables $x_2, \cdots, x_{j-1}$.

In step 7, it costs $\mathcal{O}\left(\sum_{i=0}^{d\sigma} s_i \log(s_i) + \sum_{i=0}^{d\tau} t_i \log(t_i)\right)$ number of comparisons to sort the monomial evaluations and search for identical values along the sorted arrays. This is $\mathcal{O}(\log(s)(\#\sigma_f + \#\tau_f))$.

For step 10, monomial evaluations and its coefficients are stored in two arrays, say M and C. At the first iteration, each entry in M is squared and then multiplied by the corresponding coefficient in C to compute the sum. Each iteration costs $3(\#u + \#w + \#c)$ arithmetic operations. The total cost is $\mathcal{O}(s(\#f_{j-1} + \#g_{j-1} + \#a_j))$.

In step 12, each univariate Diophantine solver costs $\mathcal{O}(d_1^2)$.

In step 14 to 16, the Vandermonde solver costs $\sum_{i=0}^{d\sigma} \mathcal{O}(s_i^2) \subseteq \mathcal{O}(s\#\sigma_f)$.

Assume $j - 2 \lessapprox s$, $\#f_{j-1} \leq \#a_j$ and $\#g_{j-1} \leq \#a_j$. We also have $\#\sigma_f \leq \#f_{j-1}$ and $\#\tau_f \leq \#g_{j-1}$. The total cost of Algorithm 2 is

$$\mathcal{O}(\underbrace{s(\#f_{j-1} + \#g_{j-1} + \#a_j)}_{\text{Eval in line 10}} + \underbrace{s\, d_1^2}_{\text{line 12}} + \underbrace{s(\#\sigma_f + \#\tau_f)}_{\text{Solve in line 14-16}}) \subseteq \mathcal{O}(s(\#a_j + d_1^2)). \square$$

Appendix B Proof of Theorem 4

Similar to the analysis of Algorithm 2, we bound the total number of arithmetic operations in $\mathbb{Z}_p$ for the worst case in Algorithm 4. Let $d_{\max} = \max_{i=2}^{j-1} \deg(a, x_i)$.

The total cost of evaluations in step 5 is $\mathcal{O}((j-2)(\#f_{j-1} + \#g_{j-1} + d_{\max}))$.

The **if** statement in step 6 costs $\mathcal{O}\left(\sum_{i=0}^{df-1} s_i \log(s_i) + \sum_{i=0}^{dg-1} t_i \log(t_i)\right) \subseteq \mathcal{O}(\log(s)(\#f_{j-1} + \#g_{j-1}))$ comparisons to sort and search for identical values.

The total cost of step 10 is $\mathcal{O}(s(\#f_{j-1} + \#g_{j-1} + \#a_j))$.

Each bivariate Hensel lift in line 12 costs $\Theta(d_1 d_j^2 + d_j d_1^2)$ [14].

Using Zippel [22] the total cost of the Vandermonde solver in step 17 is $\sum_{i=0}^{df-1} d_j \mathcal{O}(s_i^2) \subseteq \mathcal{O}(d_j s\#f_{j-1})$ for f_j. Similarly, for g_j, we have $\mathcal{O}(d_j s\#g_{j-1})$.

Assume $j - 2 \lessapprox s$, the total cost of Algorithm 4 is

$$\mathcal{O}(\underbrace{s(\#f_{j-1} + \#g_{j-1} + \#a_j)}_{\text{Eval in line 10}} + \underbrace{s(d_1^2 d_j + d_1 d_j^2)}_{\text{BHL in line 12}} + \underbrace{s\, d_j(\#f_{j-1} + \#g_{j-1})}_{\text{Solve in line 17}})$$
$$\subseteq \mathcal{O}(d_j s(\#f_{j-1} + \#g_{j-1} + d_1^2 + d_1 d_j) + s\#a_j). \square$$

References

1. Bernardin, L. and Monagan, M.: Efficient multivariate factorization over finite fields. In Proceedings of AAECC '97, Springer-Verlag LNCS **1255**:15–28 (1997)
2. Diaz A., and Kaltofen E.: FOXBOX: A System for Manipulating Symbolic Objects in Black Box Representation In Proceedings of ISSAC '98, pp. 30–37, ACM (1998)
3. Frigo M., Leiserson C.E., and Randall K.H.: The Implementation of the Cilk-5 Multithreaded Language. In Proceedings of PLDI 1998, pp. 212–223, ACM (1998)
4. Von zur Gathen, J. and Kaltofen, E.: Factorization of multivariate polynomials over finite fields. Math. Comp. **45**(7):251–261 (1985)
5. Geddes, K.O., Czapor, S.R. and Labahn, G.: Algorithms for Computer Algebra. Kluwer Acad. Publ (1992)
6. Kaltofen, E.: Sparse Hensel lifting. In Proceedings of EUROCAL '85, Springer LNCS **204**:4–17 (1985)
7. Kaltofen E., Trager, B.M.: Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Cmpt.* **9**(3): 301–320, Elsevier (1990)
8. Lang, S.: Fundamentals of Diophantine Geometry. Springer Science+Business Media New York (1983)
9. Monagan, M., Tuncer, B.: Using Sparse Interpolation in Hensel Lifting. In Proceedings of CASC 2016, LNCS **9890**, pp. 381–400, Springer (2016)
10. Monagan, M., Tuncer, B.: Factoring multivariate polynomials with many factors and huge coefficients. In Proceedings of CASC 2018, LNCS **11077**:319–334, Springer (2018)
11. Monagan, M., Tuncer, B.: Sparse multivariate Hensel lifting: a high-performance design and implementation. In Proceedings of ICMS 2018, LNCS **10931**:359–368, Springer (2018)
12. Monagan, M., Tuncer, B.: The complexity of sparse Hensel lifting and sparse polynomial factorization. *J. Symb. Cmpt.* **99**: 189–230, Elsevier (2020)
13. Monagan, M., Tuncer, B.: Polynomial Factorization in Maple 2019. In *Maple in Mathematics Education and Research.* Communications in Computer and Information Science, **1125**:341–345, Springer, 2020.
14. Monagan, M.: Linear Hensel Lifting for $\mathbb{F}_p[x,y]$ and $\mathbb{Z}[x]$ with Cubic Cost. In Proceedings of ISSAC 2019, pp. 299–306, ACM Digital Library (2019)
15. Javadi, S.M.M. and Monagan, M.: On factorization of multivariate polynomials over algebraic number and function fields. In Proceedings of ISSAC '09, pp. 199–206, ACM Digital Library (2009)
16. Lee, M.M.: Factorization of multivariate polynomials, PhD Thesis. (2013)
17. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM, **27**:701–717 (1980)
18. Wang, P.S., Rothschild, L.P.: Factoring multivariate polynomials over the integers. Mathematics of Computations, **29**:935–950 (1975)
19. Yun, D.Y.Y.: The Hensel Lemma in algebraic manipulation. Ph.D. Thesis (1974)
20. Zippel, R.E.: Probabilistic algorithms for sparse polynomials. In Proc. EUROSAM '79, Springer LNCS **72**:216–226 (1979)
21. Zippel, R.E.: Newton's iteration and the sparse Hensel algorithm. In Proc. ACM Symposium on Symbolic Algebraic Computation, pp. 68–72 (1981)
22. Zippel, R.E.: Interpolating polynomials from their values. J. Symbolic Comput. **9**(3):375–403, Elsevier (1990)
23. Zhi, L.: Optimal algorithm for algebraic factoring. Computer Science and Technology, **12**(1):1–9 (1996)