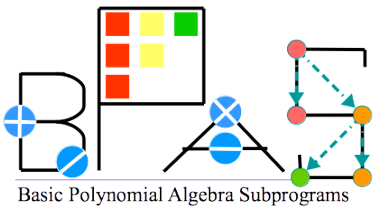
On the Complexity and Parallel Implementation of Hensel's Lemma and Weierstrass Preparation

Alexander Brandt, Marc Moreno Maza

Ontario Research Center for Computer Algebra
Department of Computer Science
University of Western Ontario, Canada

September 17, 2021
CASC 2021



ORCCA
Ontario Research Centre for Computer Algebra

The Problem

Given a polynomial $f \in \mathbb{K}[X_1, \ldots, X_n, Y]$ compute the roots of f in Y.

- → The classical Newton–Puiseux method is known to compute the roots of $f \in \mathbb{K}[X_1, Y]$ in Y as Puiseux series in X
- → The **Hensel–Sasaki Construction** or **Extended Hensel Construction** supports multivariate polynomials, computing the roots of f in Y as multivariate Puiseux series [12]
- → If f is monic, roots in Y can be computed as multivariate *power series*; we call such a method **Hensel Factorization**

Applications:

- → Computing limits of multivariate rational functions [2]
- → Computing topological closures, limit points of quasi-components [1]
- → An effective variant of the Abhyankar-Jung Theorem

Goals and Previous Work

Our goal is to improve the practical performance of the Hensel–Sasaki Construction and its applications.

→ First step is improving the monic case of Hensel factorization.

Practical efficiency:

→ High-performance and memory-efficient underlying polynomial arithmetic [4]

→ Lazy power series supporting lazy Weierstrass preparation and lazy Hensel factorization [6]

→ Now: **parallel and concurrent processing**

→ Now: dynamic load-balancing using **complexity estimates**

In This Work

- → Examine the complexity of a lazy scheme for Weierstrass Preparation Theorem and Hensel factorization
- → Present parallel algorithms for WPT and Hensel factorization
- → Composition of parallel schemes, WPT within Hensel factorization
- → Application of complexity estimates to dynamically load-balance parallel algorithm
- → Implementation of parallel algorithms supporting arbitrary number of variables in the BPAS library [3]
- → Experimental evaluation of parallel performance

Outline

Notations

$\mathbb{A} = \mathbb{K}[[X_1, \ldots, X_n]]$ is the ring of multivariate power series over an algebraically closed field. Its maximal ideal is $\mathcal{M} = \langle X_1, \ldots, X_n \rangle$.

- $f = \sum_{e \in \mathbb{N}^n} a_e X^e \in \mathbb{K}[[X_1, \ldots, X_n]]$
- $X^e = X_1^{e_1} \cdots X_n^{e_n}, \quad |e| = e_1 + \cdots + e_n$
- $f_{(k)} = \sum_{|e|=k} a_e X^e$ is the **homogeneous part** of f of degree k
- $f_{(k)} \in \mathcal{M}^k \setminus \mathcal{M}^{k+1}$
- The units of $\mathbb{A}$ are $\{u \,|\, u \notin \mathcal{M}\}$

$\mathbb{A}[Y]$ is the ring of Univariate Polynomials over Power Series (UPoPS)

- $f = \sum_{i=0}^{d} a_i Y^i, \; a_i \in \mathbb{A}, \; a_d \neq 0$ is a UPoPS of degree d
- Denote degree of f in Y by $\deg(f) = d$

Lazy Power Series

Power series implemented using a *lazy evaluation* scheme allow for terms to be computed on demand, increasing **precision** as needed.

Our lazy power series require:

1. an **update function** to compute homogeneous parts of a given degree
2. capturing parameters required for the update function
3. storing previously computed homogeneous parts

Where update parameters are power series, they are called **ancestors**.

Addition, $f = g + h$

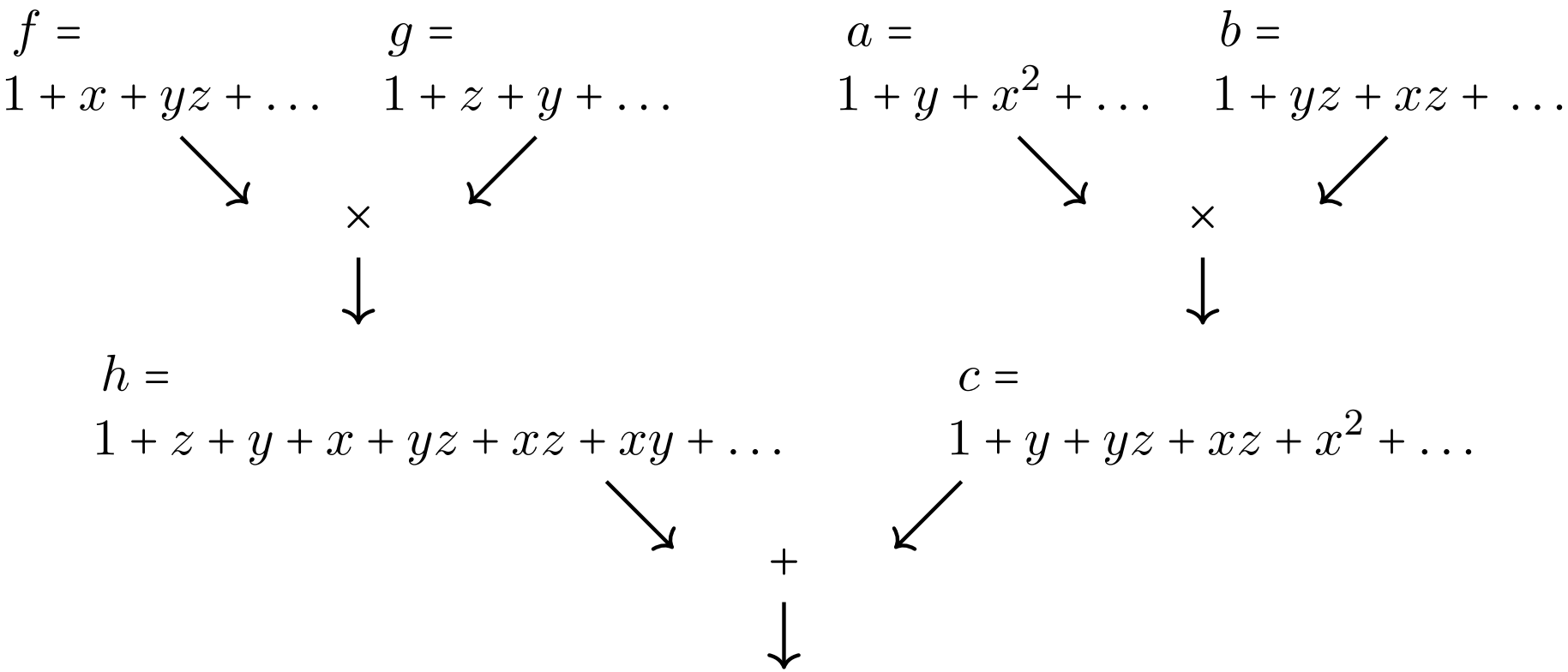
$\rightarrow \; f_{(k)} = g_{(k)} + h_{(k)}$

Multiplication $f = gh$

$\rightarrow \; f_{(k)} = \sum_{i=0}^{k} g_{(i)} h_{(k-i)}$

Ancestry Example

$$p = fg + ab$$



Outline

Weierstrass Preparation Theorem for UPoPS

Theorem 1 (Weierstrass Preparation)

Let $f = \sum_{i=0}^{d+m} a_i Y^i \in \mathbb{K}[[X_1, \ldots, X_n]][Y]$ *where* $d \geq 0$ *be the smallest integer such that* $a_d \notin \mathcal{M}$ *and* $m \in \mathbb{Z}^+$. *Assume* $f \not\equiv 0 \mod \mathcal{M}[Y]$. *Then, there exists a unique pair* p, α *satisfying the following:*

1. $f = p\,\alpha$,
2. α *is an invertible element of* $\mathbb{K}[[X_1, \ldots, X_n]][[Y]]$,
3. p *is a monic polynomial of degree* d,
4. *writing* $p = Y^d + b_{d-1}Y^{d-1} + \cdots b_1 Y + b_0$, *we have* $b_{d-1}, \ldots, b_0 \in \mathcal{M}$.

Theorem 2 (Lazy Weierstrass Complexity)

For $f = p\,\alpha \in \mathbb{K}[[X_1]][Y], \deg(p) = d, \deg(\alpha) = m$, *computing* p *and* α *to precision* k *requires* $d(m+1)k^2 + dmk$ *operations in* $\mathbb{K}$.

Lazy Weierstrass Preparation

Let $f = \sum_{\ell=0}^{d+m} a_\ell Y^\ell$, $p = Y^d + \sum_{j=0}^{d-1} b_j Y^j$, $\alpha = \sum_{i=0}^{m} c_i Y^i$ be UPoPS.

↳ a_ℓ, b_j, c_i are power series ↳ $b_j \in \mathcal{M}$ for $j = 0, \ldots, d-1$

$$
f = p\,\alpha \implies
\begin{aligned}
a_0 &= b_0 c_0 \\
a_1 &= b_0 c_1 + b_1 c_0 \\
&\vdots \\
a_{d-1} &= b_0 c_{d-1} + b_1 c_{d-2} + \cdots + b_{d-2} c_1 + b_{d-1} c_0 \\
a_d &= b_0 c_d + b_1 c_{d-1} + \cdots + b_{d-1} c_1 + c_0 \\
&\vdots \\
a_{d+m-1} &= b_{d-1} c_m + c_{m-1} \\
a_{d+m} &= c_m
\end{aligned}
$$

We update p and α by solving these equations modulo $\mathcal{M}^k$, $k = 1, 2, \ldots$

modulo $\mathcal{M}$ we have:

(1) $b_j \equiv 0 \mod \mathcal{M}$, $j = 0, \ldots, d-1$ (2) $c_i \equiv a_{d+i} \mod \mathcal{M}$ $i = 0, \ldots, m$

Lazy Weierstrass Phase 1: update p

Let $f = \sum_{\ell=0}^{d+m} a_\ell Y^\ell$, $p = Y^d + \sum_{j=0}^{d-1} b_j Y^j$, $\alpha = \sum_{i=0}^{m} c_i Y^i$ be UPoPS.

↳ a_ℓ, b_j, c_i are power series

↳ $b_j \in \mathcal{M}$ for $j = 0, \ldots, d-1$

$$\begin{aligned} a_0 &= b_0 c_0 \\ a_1 - b_0 c_1 &= b_1 c_0 \\ a_2 - b_0 c_2 - b_1 c_1 &= b_2 c_0 \\ &\vdots \\ a_{d-1} - b_0 c_{d-1} - b_1 c_{d-2} + \cdots - b_{d-2} c_1 &= b_{d-1} c_0 \end{aligned}$$

→ let $F_j = a_j - \sum_{i=0}^{j-1} b_i c_{j-i}$

→ with power series division we can solve $F_j = b_j c_0$ from $j = 0$ to $d-1$

↳ $b_{j\,(k)} = {}^{1}\!/_{c_{0\,(0)}} \left(F_{j\,(k)} - \sum_{i=1}^{k-1} b_{j\,(i)} c_{0\,(k-i)} \right)$

→ each F_j lazily updated through lazy power series arithmetic

Lazy Weierstrass Phase 2: update α

Let $f = \sum_{\ell=0}^{d+m} a_\ell Y^\ell$, $p = Y^d + \sum_{j=0}^{d-1} b_j Y^j$, $\alpha = \sum_{i=0}^{m} c_i Y^i$ be UPoPS.

↳ a_ℓ, b_j, c_i are power series

↳ $b_j \in \mathcal{M}$ for $j = 0, \ldots, d-1$

$$\begin{aligned} c_m &= a_{d+m} \\ c_{m-1} &= a_{d+m-1} - b_{d-1}c_m \\ c_{m-2} &= a_{d+m-2} - b_{d-2}c_m - b_{d-1}c_{m-1} \\ &\vdots \\ c_0 &= a_d - b_0 c_d - b_1 c_{d-1} - \cdots - b_{d-1}c_1 \end{aligned}$$

→ $c_{m-i(k)} = a_{d+m-i(k)} - \sum_{j=1}^{i} \left(b_{d-j}c_{m-i+j}\right)_{(k)}$, for $i \leq d$

→ each c_{m-i} lazily updated through lazy power series arithmetic

→ $\left(b_{d-j}c_{m-i+j}\right)_{(k)}$ only requires up to $c_{m-i+j\,(k-1)}$ since $b_{d-j\,(0)} = 0$

→ each c_{m-i} can thus be updated simultaneously

Parallel Opportunities in Weierstrass

parallel map-reduce or **parallel for** loops

In phase 1: $b_{j(k)} = {}^{1}/_{c_{0(0)}} \left(F_{j(k)} - \sum_{i=1}^{k-1} b_{j(i)} c_{0(k-i)} \right)$

→ compute summations using map-reduce:

$$\sum_{i=1}^{k} b_{j(i)} c_{0(k-i)}$$

→ notice in multivariate case, e.g., $b_{j(1)} c_{0(k-1)}$ is less work than $b_{j(\frac{k}{2})} c_{0(k-\frac{k}{2})}$

In phase 2: $c_{m-i(k)} = a_{d+m-i(k)} - \sum_{j=1}^{i} \left(b_{d-j} c_{m-i+j} \right)_{(k)}$

→ compute each $c_{m-i(k)}$, $0 \leq i \leq m$ simultaneously, and/or

→ compute product homogeneous parts using a map-reduce:

$$\left(b_{d-j} c_{m-i+j} \right)_{(k)} = \sum_{\ell=1}^{k} b_{d-j(\ell)} c_{m-i+j(k-\ell)}$$

→ load-balance issue again in the multivariate case

Update To Deg: Map-Reduce

Algorithm 1 UpdateToDegParallel$(k,\ f,\ t)$

Input: $k \in \mathbb{Z}^+$, $f \in \mathbb{K}[[X_1, \ldots, X_n]]$ known to at least precision $k-1$. If f has ancestors, it is the result of a binary operation. $t \in \mathbb{Z}^+$ for the number of threads to use.

Output: f is updated to precision k, in place.

1: $g,\ h :=$ FirstAncestor(f), SecondAncestor(f)
2: UpdateToDegParallel$(k,\ g,\ t)$;
3: UpdateToDegParallel$(k,\ h,\ t)$;

4: **if** f is a product **then** ▷ compute $f_{(k)}$ by map-reduce
5: | $\mathcal{V} := [0, \ldots, 0]$ ▷ 0-indexed list of size t
6: | **parallel_for** $j := 0$ to $t-1$
7: | | **for** $i := {}^{jk}/_{t}$ to ${}^{(j+1)k}/_{t} - 1$ **while** $i \leq k$ **do**
8: | | | $\mathcal{V}[j] := \mathcal{V}[j] + g_{(i)} h_{(k-i)}$
9: | $f_{(k)} := \sum_{j=0}^{t-1} \mathcal{V}[j]$ ▷ reduce

...

Parallel Challenges

$$
\begin{aligned}
c_m &= a_{d+m} \\
c_{m-1} &= a_{d+m-1} - b_{d-1}c_m \\
c_{m-2} &= a_{d+m-2} - b_{d-2}c_m - b_{d-1}c_{m-1} \\
c_{m-3} &= a_{d+m-3} - b_{d-3}c_m - b_{d-2}c_{m-1} - b_{d-1}c_{m-2} \\
&\vdots \\
c_0 &= a_d - b_0c_d - b_1c_{d-1} - \cdots - b_{d-1}c_1
\end{aligned}
$$

- → Computing $c_{m-i(k)}$ for different i requires a different amount of work
- → c_{m-i} requires i products
- → For load-balance, must compute a different number of c_{m-i} with each thread; the total number of products computed per thread should be same

Outline

Hensel's Lemma

Theorem 3 (Hensel's Lemma)

Let $f = Y^d + \sum_{i=0}^{d-1} a_i Y^i$ *be a monic polynomial in* $\mathbb{K}[[X_1, \ldots, X_n]][Y]$. *Let* $\bar{f} = f(0, \ldots, 0, Y) = (Y - c_1)^{d_1}(Y - c_2)^{d_2} \cdots (Y - c_r)^{d_r}$ *for* $c_1, \ldots, c_r \in \mathbb{K}$ *and positive integers* $d_1, \ldots, d_r$. *Then, there exists* $f_1, \ldots, f_r \in \mathbb{K}[[X_1, \ldots, X_n]][Y]$, *all monic in Y, such that:*

1. $f = f_1 \cdots f_r$,
2. $\deg(f_i, Y) = d_i$ *for* $1 \leq i \leq r$, *and*
3. $\bar{f}_i = (Y - c_i)^{d_i}$ *for* $1 \leq i \leq r$.

Proof:

Let $g = f(X_1, \ldots, X_n, Y + c_r) = Y^d + \sum_{i=0}^{d-1} b_i Y^i$, sending c_r to the origin. By construction, $b_0, \ldots, b_{d_r - 1} \in \mathcal{M}$ and Weierstrass preparation can be applied to produce $g = p\,\alpha$ with $\deg(p) = d_r, \deg(\alpha) = d - d_r$.

Reversing the shift, $f_r = p(Y - c_r)$.
Induction on $\widehat{f} = \alpha(Y - c_r)$ completes the proof. □

Hensel Factorization

Algorithm 2 HenselFactorization(f)

Input: $f = Y^d + \sum_{i=0}^{d-1} a_i Y^i, a_i \in \mathbb{K}[[X_1, \ldots, X_n]]$.
Output: $f_1, \ldots, f_r$ satisfying Theorem 3.

1: $\bar{f} = f(0, \ldots, 0, Y)$
2: $(c_1, \ldots, c_r), (d_1, \ldots, d_r) :=$ roots and their multiplicities of $\bar{f}$
3: $c_1, \ldots, c_r :=$ SORT$([c_1, \ldots, c_r])$ by increasing multiplicity
4: $\widehat{f}_1 := f$
5: **for** $i := 1$ to $r - 1$ **do**
6: $\quad g_i := \widehat{f}_i(Y + c_i)$
7: $\quad p_i, \alpha_i :=$ WeierstrassPreparation(g_i)
8: $\quad f_i := p_i(Y - c_i)$
9: $\quad \widehat{f}_{i+1} := \alpha_i(Y - c_i)$
10: $f_r := \widehat{f}_r$
11: **return** $f_1, \ldots, f_r$

Hensel Complexity (1/2)

Let $\widehat{d_i} = \deg(\widehat{f_i}) = \sum_{j=i}^{r} d_i$

At each iteration but the last:

1. a shift of degree $\widehat{d_i}$ for $\widehat{f_i}(Y + c_i)$
2. a Weierstrass preparation producing $\deg(p) = d_i, \deg(\alpha) = \widehat{d_{i+1}}$
3. a shift of degree d_i for $p(Y - c_i)$
4. a shift of degree $\widehat{d_{i+1}}$ for $\alpha(Y - c_i)$

Theorem 4 (Hensel Factorization Complexity)

For $f = f_1 \cdots f_r \in \mathbb{K}[[X_1]][Y], \deg(f) = d$, updating all factors to precision k can be done within $\mathcal{O}(d^3k + d^2k^2)$ operations in $\mathbb{K}$.

Conjecture: using relaxed algorithms [8] in Weierstrass may yield $\mathcal{O}(d^3k + d^2k\log^2 k)$

Hensel Complexity (2/2)

Let $f \in \mathbb{K}[[X_1]][Y]$ have degree d_Y in Y and total degree d.
Let $M(n)$ be a polynomial multiplication time.

Factorizing f requires...

- → Our method: $\mathcal{O}(d_Y^3 k + d_Y^2 k^2)$
- → Hensel–Sasaki Construction (EHC): $\mathcal{O}(d^3 M(d) + k^2 d M(d))$ [2]
- → Over $\mathbb{C}$, Newton–Puiseux can be done in:
  - ↳ $\mathcal{O}(d^2 M(k))$; Kung and Traub [9]
  - ↳ $\mathcal{O}(d^2 k)$; Chudnovsky and Chudnovsky [7]
- → Berthomieu, Lecerf, Quintin: $\mathcal{O}(M(d_Y)\log(d_Y)\, k M(k))$ [5]

Including the initial root-finding cost as $R(d_Y)$:

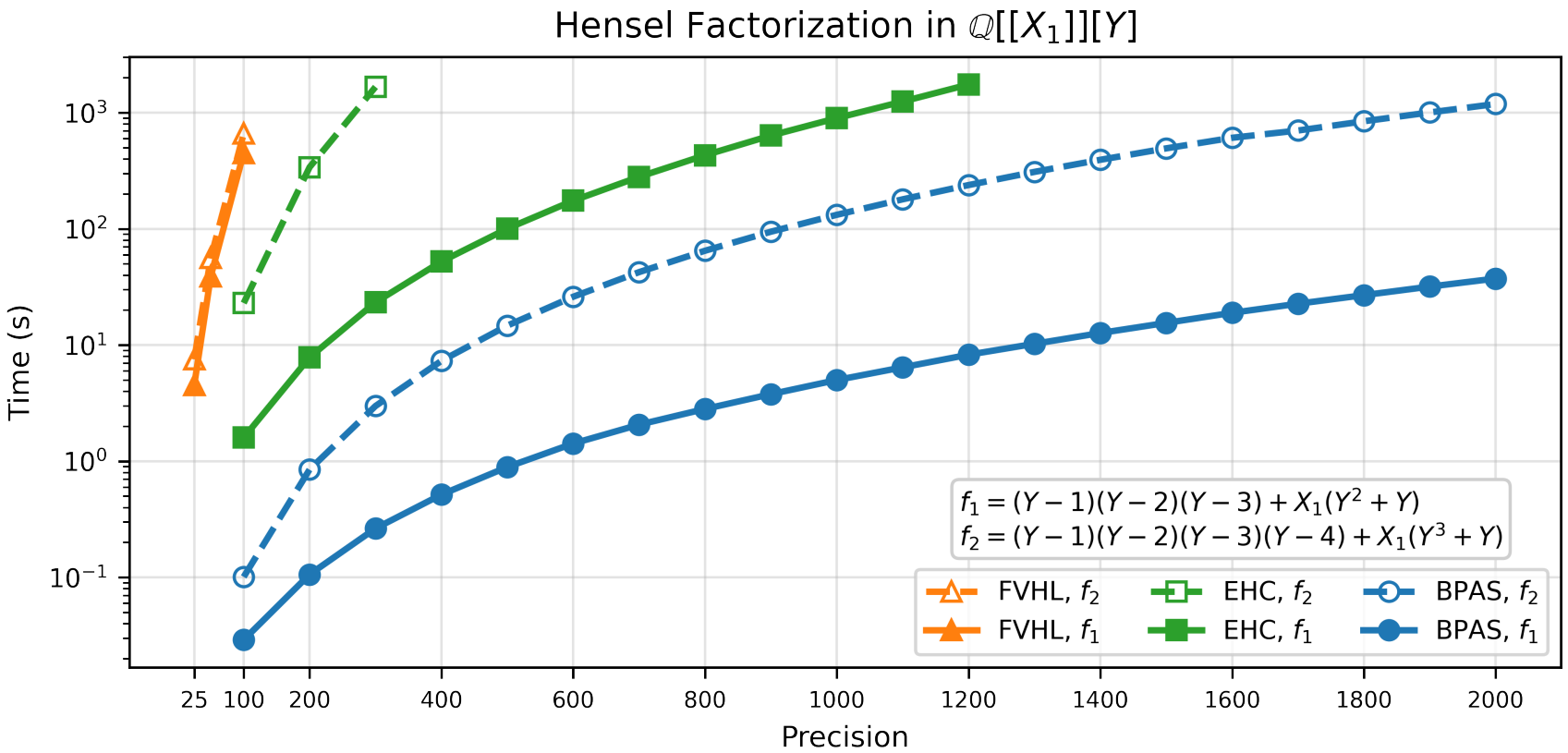
- → Our method: $\mathcal{O}(d_Y^3 k + d_Y^2 k^2 + R(d_Y))$
- → Neiger, Rosenkilde, Schost: $\mathcal{O}^{\sim}(d_Y k + k R(d_Y))$ [11]

Note $M(n) \in \mathcal{O}(n^2)$ or $\mathcal{O}(n \log n)$; $R(n) \in \mathcal{O}^{\sim}(n^2)$

Serial Performance

EHC outperforms Kung and Traub's method by orders of magnitude [2]

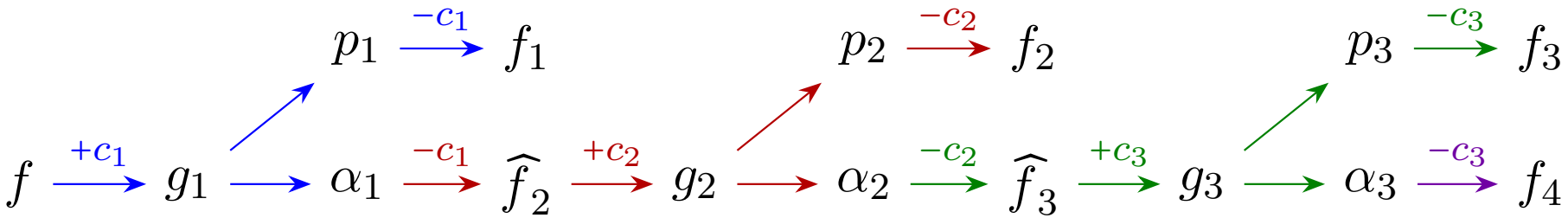
From CASC 2020, our serial method outperforms EHC and Factorization via Hensel's Lemma of `PowerSeries` sub-package of `RegularChains` [10]



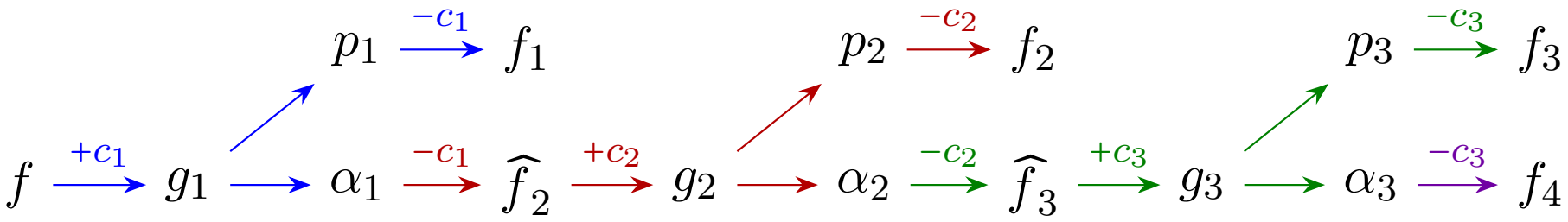
Parallel Opportunities in Hensel

- → The output of one Weierstrass becomes input to another
- → $f_{i+i_{(k)}}$ relies on $f_{i_{(k)}}$
- → Can compute $f_{i_{(k+1)}}$ and $f_{i+i_{(k)}}$ concurrently in a **pipeline**

	Stage 1 (f_1)	Stage 2 (f_2)	Stage 3 (f_3)	Stage 4 (f_4)
Time 1	$f_{1_{(1)}}$			
Time 2	$f_{1_{(2)}}$	$f_{2_{(1)}}$		
Time 3	$f_{1_{(3)}}$	$f_{2_{(2)}}$	$f_{3_{(1)}}$	
Time 4	$f_{1_{(4)}}$	$f_{2_{(3)}}$	$f_{3_{(2)}}$	$f_{4_{(1)}}$
Time 5	$f_{1_{(5)}}$	$f_{2_{(4)}}$	$f_{3_{(3)}}$	$f_{4_{(2)}}$
Time 6	$f_{1_{(6)}}$	$f_{2_{(5)}}$	$f_{3_{(4)}}$	$f_{4_{(3)}}$



Parallel Challenges and Composition



→ Degrees and computational work diminish with each stage
  ↳ $\deg(g_1) = d,\ \deg(g_2) = d - d_1,\ \deg(g_3) = d - d_1 - d_2,\ \ldots$

→ To load-balance each stage, give a decreasing number of threads to each stage to be used within Weierstrass preparation.

→ From Theorem 1: updating f_i requires $\mathcal{O}(d_i \widehat{d}_{i+1} k^2)$ operations

→ Assign t_i threads to stage i so that $d_i \widehat{d}_{i+1}/t_i$ is equal for each stage.

→ Better still, update a *group of successive factors* per stage.
  ↳ To each stage s assign factors $f_{s_1}, \ldots, f_{s_2}$ and t_s threads so that $\sum_{i=s_1}^{s_2} d_i \widehat{d}_{i+1}/t_s$ is roughly equal for each stage.

Hensel Pipeline Example

$$f = (Y-1)(Y-2)(Y-3)(Y-4) + X_1(Y^3+Y)$$
$$\bar{f}_1 = Y-1, \quad \bar{f}_2 = Y-2, \quad \bar{f}_3 = Y-3, \quad \bar{f}_4 = Y-4$$

$k = 600$

factor	serial time (s)	shift time (s)	$d_i \widehat{d}_{i+1}$	Complexity-est. threads	parallel time (s)	wait time (s)	Time-est. threads	parallel time (s)	wait time (s)
f_1	18.1989	0.0012	$1 \cdot 3$	6	4.5380	0.0000	7	3.5941	0.0000
f_2	6.6681	0.0666	$1 \cdot 2$	4	4.5566	0.8530	3	3.6105	0.6163
f_3	3.4335	0.0274	$1 \cdot 1$	1	4.5748	1.0855	0	-	-
f_4	0.0009	0.0009	$1 \cdot 0$	1	4.5750	4.5707	2	3.6257	1.4170

- → f_4 requires at least one thread so that it (the last factor) gets updated
- → work estimates based on complexity results okay, but does not account for, e.g., coefficient size or data locality.
- → can also use serial time to suggest thread assignments

Hensel Pipeline Algorithm

Algorithm 3 HenselPipeStage(k, f_i, t, gen)

Input: $k \in \mathbb{Z}^+$, f_i a UPoPS, $t \in \mathbb{Z}^+$ the number of threads to use within this stage. gen a generator for the previous pipeline stage.

Output: a sequence of integers j signalling f_i is known to precision j; the sequence ends at k.

```
p := Precision(f_i)  ▷ current precision of f_i
do
    ▷ A blocking function call until gen yields
    k' := gen()
    for j := p to k' do
        UpdateToDegParallel(j, f_i, t)
        yield j
    p := k'
while k' < k
```

Algorithm 4 HenselFactorizationPipeline(k, $\mathcal{F}$, $\mathcal{T}$)

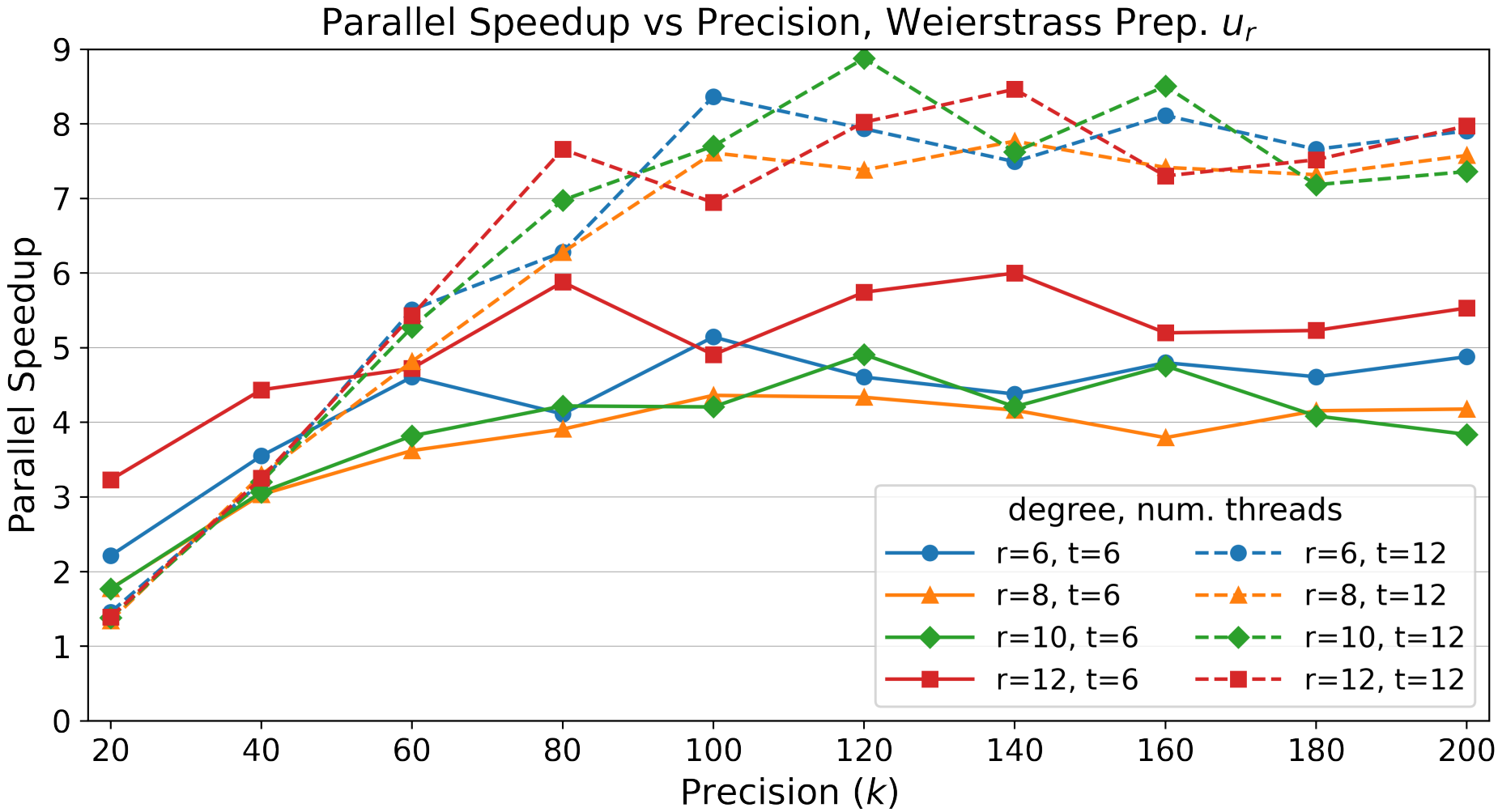
Input: $k \in \mathbb{Z}^+$, $\mathcal{F} = \{f_1, \ldots, f_r\}$ the output of HenselFactorization. $\mathcal{T} \in \mathbb{Z}^r$ a 0-indexed list of the number of threads to use in each stage, $\mathcal{T}[r-1] > 0$.

Output: $f_1, \ldots, f_r$ updated in-place to precision k.

```
▷ an anonymous function asynchronous generator
gen := ( ) → {yield k}
for i := 0 to r − 1 do
    if T[i] > 0 then
        ▷ Capture function as a function object,
          passing the previous gen as input
        gen := AsyncGenerator(
          HenselPipeStage, k, f_{i+1}, T[i], gen)
▷ ensure last stage completes before returning
do
    k' := gen()
while k' < k
```

Outline

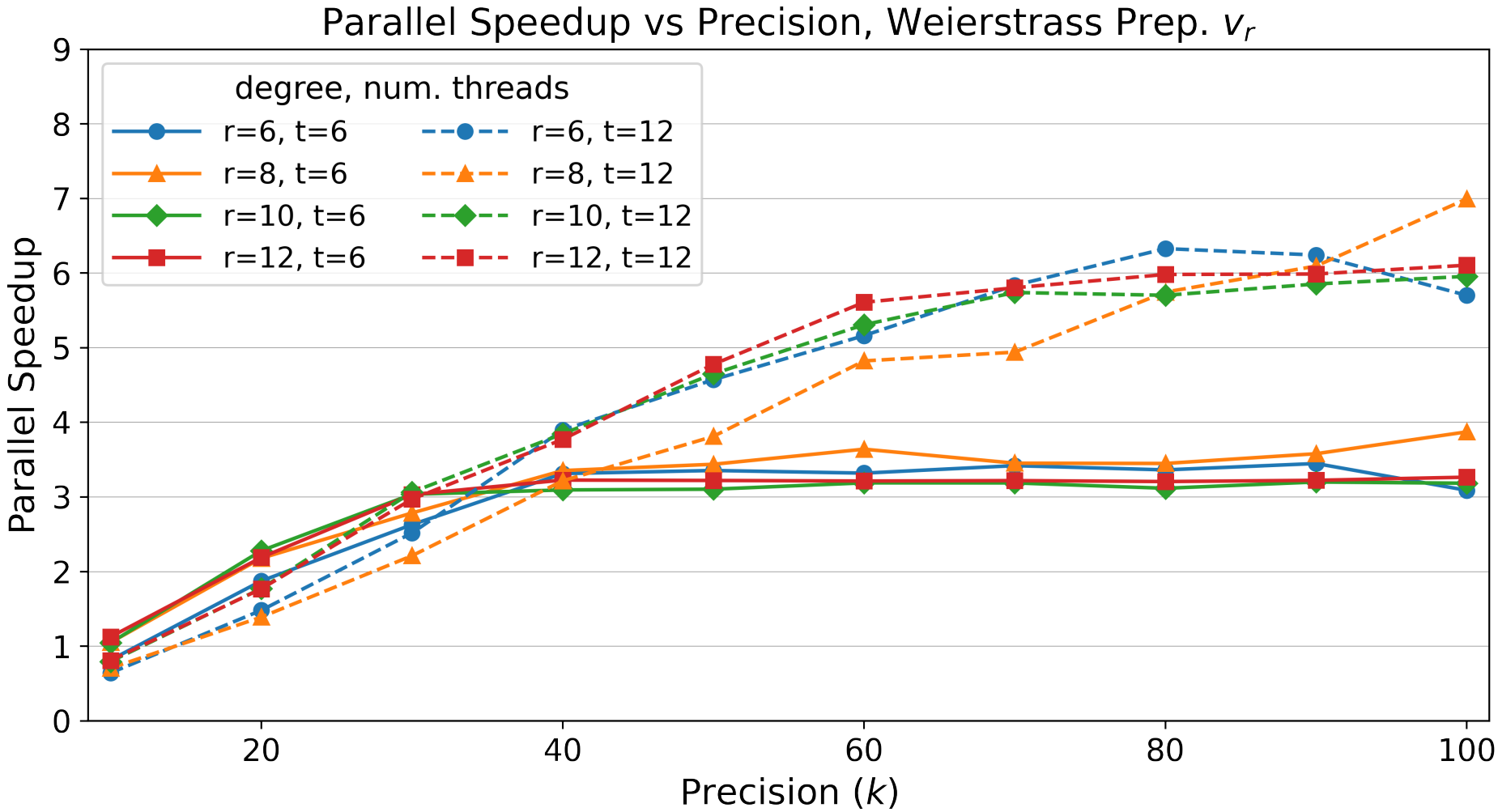
Parallel Speed-up WPT 1



$$u_r = \sum_{i=2}^{r} (X_1^2 + X_2 + i)Y^i + (X_1^2 + X_2)Y + X_1^2 + X_1X_2 + X_2^2$$

$$\implies \deg(p) = 2 \quad \deg(\alpha) = r - 2$$

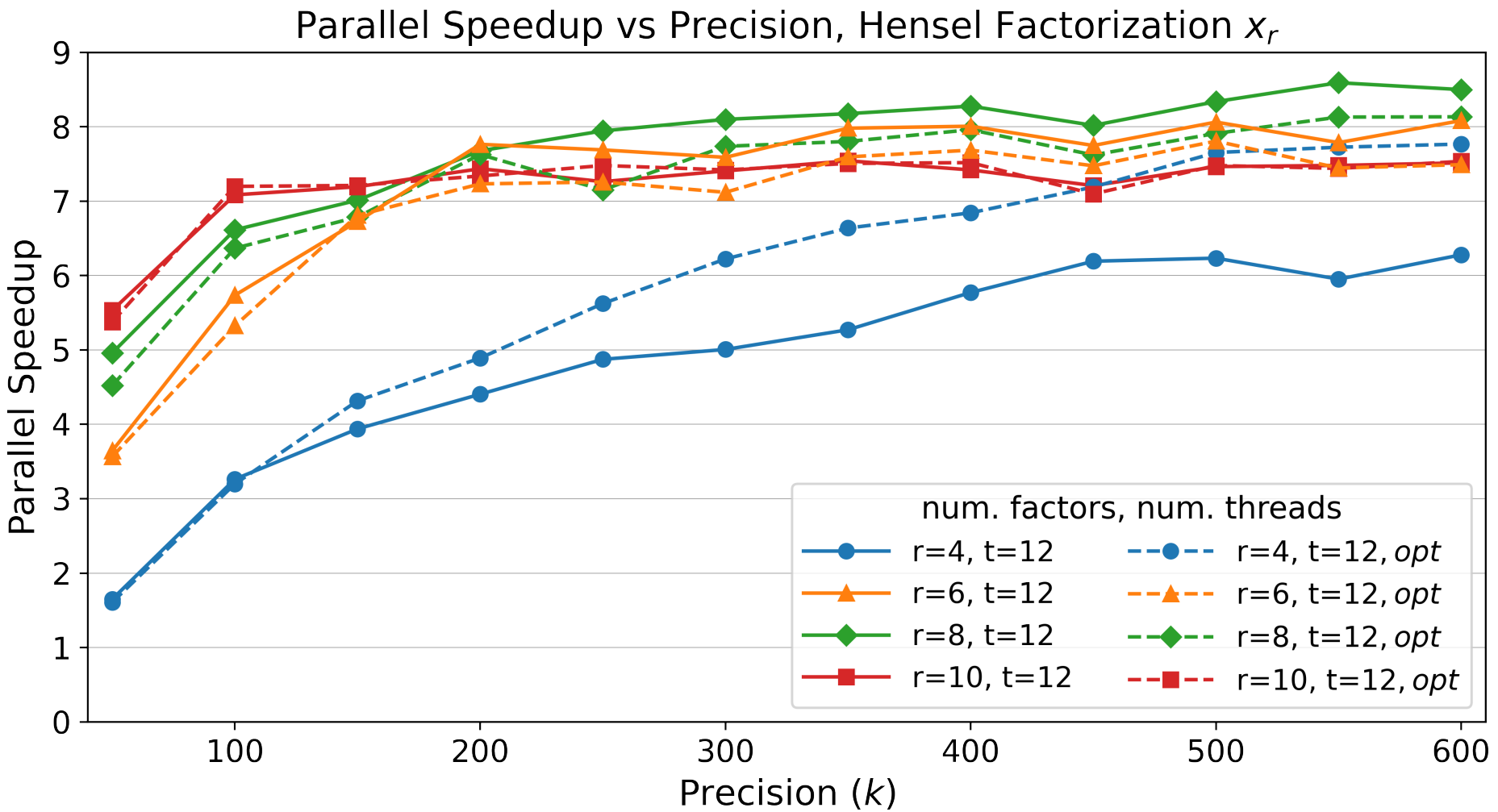
Parallel Speed-up WPT 2



$$v_r = \sum_{i=r/2}^{r} (X_1^2 + X_2 + i)Y^i + \sum_{i=1}^{r/2-1} (X_1^2 + X_2)Y^i + X_1^2 + X_1X_2 + X_2^2$$

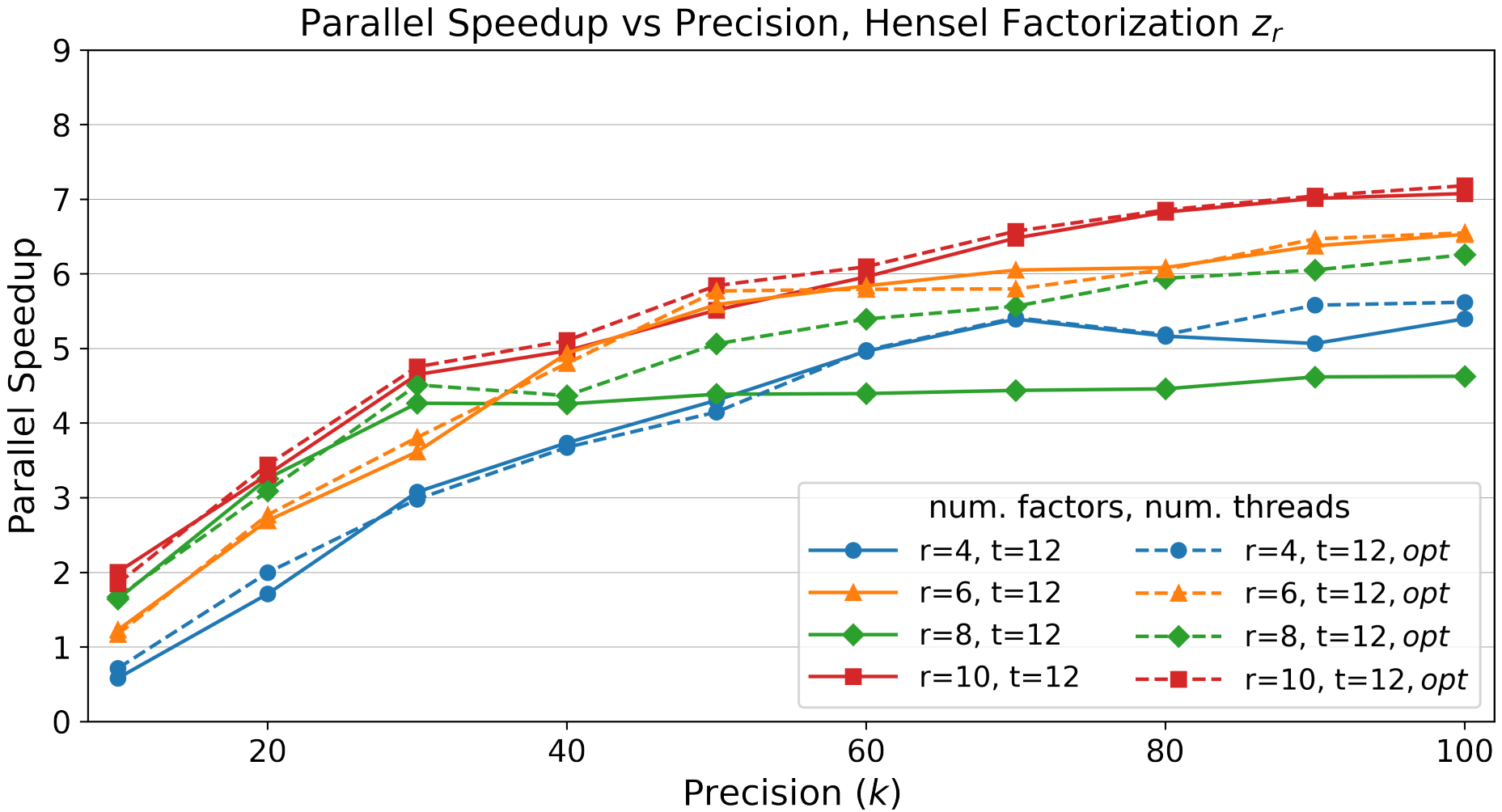
$$\Longrightarrow \deg(p) = \deg(\alpha) = r/2$$

Parallel Speed-up Hensel Factorization 1



$$x_r = \prod_{i=1}^{r} (Y - i) + X_1(Y^3 + Y)$$

Parallel Speed-up Hensel Factorization 2



$$z_r = \prod_{i=1}^{r} (Y + X_1 + X_2 - i) + X_1 X_2 (Y^3 + Y)$$

Conclusions and Future Work

→ We have seen complexity estimates and parallel algorithms for lazy Weierstrass preparation and lazy Hensel factorization.

→ The composition of parallel patterns (map-reduce within pipeline) provides good performance.

In future:

→ In multivariate case, must improve loop partitioning to better compute homogeneous parts of products: $f_{(k)} = \sum_{i=0}^{k} g_{(i)} h_{(k-i)}$

→ Improved thread distribution between Hensel pipeline stages to consider practical performance: coefficient size, locality

→ Improved thread distribution between Hensel pipeline stages for multivariate case

→ Relaxed algorithms within Weierstrass preparation

References

[1] P. Alvandi. "Computing Limit Points of Quasi-components of Regular Chains and its Applications". PhD thesis. University of Western Ontario, 2017.

[2] P. Alvandi, M. Ataei, M. Kazemi, and M. Moreno Maza. "On the Extended Hensel Construction and its application to the computation of real limit points". In: *J. Symb. Comput.* 98 (2020), pp. 120–162.

[3] M. Asadi, A. Brandt, C. Chen, S. Covanov, M. Kazemi, F. Mansouri, D. Mohajerani, R. H. C. Moir, M. Moreno Maza, D. Talaashrafi, L. Wang, N. Xie, and Y. Xie. *Basic Polynomial Algebra Subprograms (BPAS) (version 1.791)*. `http://www.bpaslib.org`. 2021.

[4] M. Asadi, A. Brandt, R. H. C. Moir, and M. Moreno Maza. "Algorithms and Data Structures for Sparse Polynomial Arithmetic". In: *Mathematics* 7.5 (2019), p. 441.

[5] J. Berthomieu, G. Lecerf, and G. Quintin. "Polynomial root finding over local rings and application to error correcting codes". In: *Appl. Algebra Eng. Commun. Comput.* 24.6 (2013), pp. 413–443.

[6] A. Brandt, M. Kazemi, and M. Moreno Maza. "Power Series Arithmetic with the BPAS Library". In: *Proc. of CASC 2020*. Vol. 12291. LNCS. Springer, 2020, pp. 108–128.

[7] D. V. Chudnovsky and G. V. Chudnovsky. "On expansion of algebraic functions in power and Puiseux series I". In: *Journal of Complexity* 2.4 (1986), pp. 271–294.

[8] J. van der Hoeven. "Relax, but Don't be Too Lazy". In: *J. Symb. Comput.* 34.6 (2002), pp. 479–542.

[9] H. T. Kung and J. F. Traub. "All Algebraic Functions Can Be Computed Fast". In: *J. ACM* 25.2 (1978), pp. 245–260.

[10] F. Lemaire, M. Moreno Maza, and Y. Xie. "The RegularChains library in MAPLE". In: *ACM SIGSAM Bulletin* 39.3 (2005), pp. 96–97.

[11] V. Neiger, J. Rosenkilde, and É. Schost. "Fast Computation of the Roots of Polynomials Over the Ring of Power Series". In: *Proc. of ISSAC 2017*. ACM, 2017, pp. 349–356.

[12] T. Sasaki and F. Kako. "Solving multivariate algebraic equation by Hensel construction". In: *Japan J. Indust. and Appl. Math.* 16.2 (1999), pp. 257–285.