Parallel Computation of the Power Series Solutions to Algebraic and Differential Equations

Greg Alejandro Solis-Reyes[1], Marc Moreno Maza[1], and Alexander Brandt[2]

[1] Ontario Research Center for Computer Algebra, Western University, Canada gsolisre@uwo.ca, moreno@csd.uwo.ca

[2] Faculty of Computer Science, Dalhousie University, Canada ABrandt@dal.ca

Abstract. We present parallel techniques developed for computing the power series solutions to certain algebraic and differential equations, along with corresponding high-performance software implementations. These specialized parallel methods reveal abstract patterns which could be generalized and applied to other problems in computational algebra.

Keywords: Power Series, Parallelization, Algebraic Equations, Differential Equations, High Performance Computer Algebra, BPAS

1 Introduction

In most parallel algebraic computations, the algorithmic patterns follow a multi-way divide-and-conquer scheme (or equivalently, *blocking* or *tiling* strategies) or a *map-reduce* [13, 14] scheme (generally a parallelizable for-loop nest). Standard examples are given by dense matrix multiplication [10], fast Fourier transforms (FFTs) [8, 20], Toom-Cook multiplication [16], and interpolation via sub-product tree techniques [11], to name a few.

In contrast, the parallelization of algorithms computing the power series solutions to algebraic and differential equations yield less common and more challenging patterns. The articles [1, 6, 7, 19] report on successful implementations of these parallel patterns. We believe that other algebraic computations could take advantage of these patterns, which motivates the present article.

Relaxed and *lazy* [12] implementations of power series allow them to be treated as computational objects and be updated on demand, in contrast with *zealous* or *truncated* approaches which deal with power series at a fixed degree.

Parallelizing the factorization of univariate polynomials over multivariate power series (UPoPS), based on the Weierstrass Preparation Theorem (WPT) and implemented via *lazy* evaluation, yields to the composition of *stencil*-like schemes and *map-reduce* schemes, see Section 2. Parallelizing the factorization of UPoPS, based on Hensel's lemma, can be done by a producer-consumer scheme (which can also be regarded as a pipeline) composed with the schemes involved in parallelizing WPT, see Section 3. When moving to the extended Hensel construction, dynamic programming takes center stage, see Section 5. Computing the power series solution to a linear ordinary differential equation (ODE) at one of its *ordinary points* can be performed by a *tiling* strategy using a *stencil* [14]. Composed with other parallel patterns, and supported by relaxed evaluation, this leads to a parallelization that achieves nearly linear speedup on fat multi-core nodes which support simultaneous multithreading (SMT) [22], see Section 4.

The algorithms presented in the articles [1, 6, 7, 19] are implemented for sequential execution in Maple's `MultivariatePowerSeries` library [21] and, for multi-threaded execution (thus as parallel algorithms) in the `BPAS` library [2], with the exception of the algorithms of [1]. `BPAS` is publically available in source at `https://bpaslib.org` and `https://github.com/orcca-uwo/BPAS`. Maple ships with the `MultivariatePowerSeries` library, and it is also publically available in source at `https://github.com/orcca-uwo/MultivariatePowerSeries`.

The `MultivariatePowerSeries` library also provides two algorithms for factoring univariate polynomials over Puiseux series, and thus for finding the Puiseux series solutions of algebraic

equations. The first is based on Nowak's proof [17] of Puiseux's theorem, which reduces the computation of these Puiseux series solutions to applying Hensel's lemma. The second one is the so-called *Extended Hensel Construction* [18] of Sasaki and Kako, which extends Puiseux's theorem to multivariate Puiseux series coefficients, discussed further in Section 5.

2 Extracting parallelism from lazy evaluation

This section uses the notations of [6, 7] and discusses the implementation of the algorithms reported in these articles using the `BPAS` library. In `BPAS`, as well as in `MultivariatePowerSeries`, the implementation of multivariate power series follow a *lazy evaluation* scheme. The encoding of a multivariate power series requires: (1) an update function to compute homogeneous parts of a given degree, (2) capturing the parameters[3] of the update function, and (3) storing previously computed homogeneous parts. Where a parameter of an update function is itself a power series, we call it an *ancestor*. We note that some of the `BPAS` operations on multivariate power series are based on relaxed algorithms rather than lazy evaluation. However, for simplicity's sake in this extended abstract, this Section focuses on lazy evaluation.

To highlight how one can extract parallelism from this lazy evaluation strategy, we consider the Weierstrass preparation theorem.

Theorem 2.1 (Weierstrass Preparation Theorem) *Let $f = \sum_{i=0}^{d+m} a_i Y^i \in \mathbb{K}[[X_1, \ldots, X_n]][Y]$ be a UPoPS where d is the smallest integer such that $a_d \notin \mathcal{M}$, m is a non-negative integer and $\mathcal{M}$ is the maximal ideal of $\mathbb{K}[[X_1, \ldots, X_n]]$. Then, there exist two UPoPS:*

$$p = Y^d + \textstyle\sum_{j=0}^{d-1} b_j Y^j \quad \text{and} \quad \alpha = \textstyle\sum_{i=0}^{m} c_i Y^i, \quad \text{such that:}$$

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

Expanding the equality $f = \alpha p$ yields the following system of polynomial equations the indeterminates of which are the power series $b_0, \ldots, b_{d-1}, c_0, \ldots, c_m$:

$$\begin{aligned}
a_0 &= b_0c_0\\
a_1 &= b_0c_1 + b_1c_0\\
a_2 &= b_0c_2 + b_1c_1 + b_2c_0\\
&\vdots\\
a_{d-1} &= b_0c_{d-1} + b_1c_{d-2} + \cdots + b_{d-2}c_1 + b_{d-1}c_0\\
a_d &= b_0c_d + b_1c_{d-1} + \cdots + b_{d-1}c_1 + c_0\\
a_{d+1} &= b_0c_{d+1} + b_1c_d + \cdots + b_{d-1}c_2 + c_1\\
&\vdots\\
a_{d+m-3} &= b_{d-3}c_m + b_{d-2}c_{m-1} + b_{d-3}c_{m-2} + c_{m-3}\\
a_{d+m-2} &= b_{d-2}c_m + b_{d-1}c_{m-1} + c_{m-2}\\
a_{d+m-1} &= b_{d-1}c_m + c_{m-1}\\
a_{d+m} &= c_m.
\end{aligned}$$

Because adding, multiplying and inverting multivariate power series are continuous operations in the Krull topology, the same is true for the computation of the power series $b_0, \ldots, b_{d-1}, c_0, \ldots, c_m$ in WPT, see [15]. In other words, the UPoPS p and α can be computed by solving the above equations modulo $\mathcal{M}^k$, for increasing values of $k = 1, 2, \ldots$. Let us start with $k = 1$. Modulo $\mathcal{M}$ we have:

$$b_j \equiv 0 \mod \mathcal{M},\ j = 0, \ldots, d-1 \text{ then } c_i \equiv a_{d+i} \mod \mathcal{M} \text{ for } i = 0, \ldots, m.$$

[3] Here, the word *parameter* should be understood in the sense of the `C` programming language, which is used for `BPAS`.

Assume now that $b_0, \ldots, b_{d-1}, c_0, \ldots, c_m$ have been computed modulo $\mathcal{M}^k$, for some value of k, and that we want to update them so that their homogeneous parts in degree k become known. We first update $b_0, \ldots, b_{d-1}$. To do so, we define

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

and observe that with power series division we can solve $F_j = b_j c_0$ from $j = 0$ to $j = d-1$. To be precise, the homogeneous part $b_{j\,(k)}$ of b_j in degree k is given by:

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

While the homogeneous parts $b_{j\,(k)}$ are computed one after another, parallelism is extracted from: (1) updating concurrently all the ancestors of $a_0, \ldots, a_{d+m}$, (2) computing the sums $\sum_{i=1}^{k} b_{j\,(i)} c_{0\,(k-i)}$ in a map-reduce fashion.

Next, we update $c_0, \ldots, c_m$. We observe that for $0 \leq i \leq m$ we have:

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

Parallelism is extracted from: (1) the fact all $c_{m-i\,(k)}$'s can be computed simultaneously, and (2) applying map-reduce to evaluate:

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

Clearly, there are implementation challenges in the above scheme. In particular, load-balancing in the evaluation of sums of products requires fine tuning in the multivariate case.

The experimental results of [7] show that on an 12-core node, the parallel computaton of the UPOPS p and α, as described above, achieves a speedup factor of 9 for sufficiently large values of k.

3 Hensel lifting in a pipeline

Another factorization result to which the techniques of Section 2 apply is Hensel's lemma. To be precise, the factors produced by Hensel's lemma can be computed by composing the scheme of Section 2 for WPT with a pipelined computation. To see this, we recall Hensel's lemma and a proof of it relying on WPT.

Theorem 3.1 (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) $f = f_1 \cdots f_r$ holds, (2) we have $\deg(f_i, Y) = d_i$ for $1 \leq i \leq r$, and (3) $\bar{f}_i = (Y - c_i)^{d_i}$ holds, 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, we have $b_0, \ldots, b_{d_r - 1} \in \mathcal{M}$ and WPT can be applied to produce $g = p\,\alpha$ with $\deg(p) = d_r, \deg(\alpha) = d - d_r$. Reversing the shift, we have $f_r = p(Y - c_r)$. Induction on $\hat{f} = \alpha(Y - c_r)$ completes the proof.

Figure 1 illustrates the proof for the case $r = 4$.

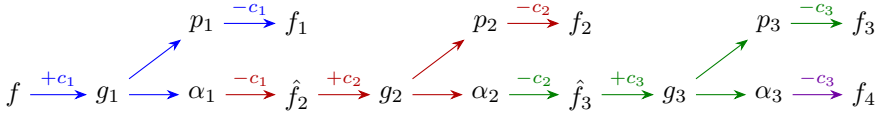

Fig. (1) Depiction of a 4-stage pipeline executing Hensel's lemma.

We observe that, in this constructive proof, the output of one application of WPT provides the input to another such application of WPT. As a result, following the algorithm induced by this proof, the computation of the homogeneous part $f_{i+1(k)}$ of f_{i+1} in degree k relies on the computation of the homogeneous part $f_{i(k)}$ of f_i in degree k. Therefore the homogeneous parts $f_{i(k+1)}$ and $f_{i+1(k)}$ can be computed concurrently in a pipeline. Table 1 illustrates this opportunity for concurrent execution in the case $r = 4$.

The experimental results of [7] show that on an 12-core node, parallel computation of the factors of Hensel lemma factorizations modulo $\mathcal{M}^k$, as described above, achieves a speedup factor of 9 for sufficiently large values of k.

	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	$f_{1(6)}$	$f_{2(5)}$	$f_{3(4)}$	$f_{4(3)}$

Table (1) Simulation of a 4-stage pipeline executing Hensel's lemma.

4 Power series solutions to ODEs via stencil computations

Computing the power series solution of an order d linear ODE with formal power series coefficients subject to d initial conditions is a classical problem. Such an ODE can be written as $\sum_{i=0}^{d} a_i \cdot y^{(i)} = b$ where $a_i, b, y \in \mathbb{Q}[[x]]$, $a_i = \sum_n a_{i,n} x^n$, $b = \sum_n b_n x^n$, $y = \sum_n c_n x^n$. This problem has been studied in [19] and [5] for ODEs where a_d is invertible ($a_{d,0} \neq 0$). In this Section the notations of [19] are used and complexity is measured in arithmetic operations.

In [5], the authors describe an algorithm providing quasi-linear asymptotic complexity through the use of polynomial multiplication in a *zealous* divide-and-conquer approach. The algorithm has $O(dM(k) \log k)$ complexity, where $M(k)$ is the complexity of polynomial multiplication of degree k. In [19], the focus is on parallelization and a *relaxed* implementation of a quadratic complexity ($O(dk^2)$) algorithm through the use of a *tiling* strategy. It is known [9, Ch. 9.7] that in a practical implementation, below some *crossover point*, quadratic polynomial multiplication outperforms asymptotically faster methods. Therefore the $\log k$ factor in the complexity of the algorithm in [5] implies that below that crossover point, the $O(dk^2)$ algorithm of [19] will save that $\log k$ factor, with additional speedup from parallelization. Therefore the work in [19] can be seen to provide an algorithm suitable for *hybrid* systems.

The rest of this section follows the work of [19], highlighting parallel patterns and extending results. First, a *recurrence relation* is derived for the terms of the power series solution to a linear ODE. This "infinite order" recurrence naturally lends itself to a quadratic *lazy* algorithm through directly implementing the recurrence. Some simple dynamic programming (tabulation), and algebraic optimizations of the recurrence are applied to get a form which is suitable for parallelization. Equation (1) shows this optimized recurrence, where $r_{j,n} = \frac{(j+n)!}{n!}$, and $w_k = r_{d,k} \cdot a_{d,0}$.

$$c_{d+k} = \frac{1}{w_k} \cdot \left(b_k - \sum_{i=0}^{d+k-1} \left(c_i \sum_{m=0}^{d} \begin{cases} r_{m,i-m} \cdot a_{m,k-i+m} & \textbf{if } i \geq m \textbf{ and } k \geq i-m \\ 0 & \textbf{otherwise.} \end{cases} \right) \right) \tag{1}$$

The problem is thus to compute many of these c_{d+k} terms quickly and efficiently. The challenge in parallelizing the recurrence (1) comes from the dependence on previous terms. This is related to the famous parallel prefix sum problem [4], but is distinctly different due to the

increasing order of the recurrence. In order to apply (1) exactly, we compute over $\mathbb{Q}$, thus in characteristic zero. Therefore another challenge in the implementation and parallelization comes from the bit-size growth of the c_{d+k} terms. This growth comes at a minimum from factorial-like terms ($r_{m,i-m}$) arising from the differential operation, and may also come from growth in the terms of the coefficient power series. Here and in [19], parallel algorithms are designed and analyzed in the *fork-join model* [14].

An obvious first attempt to parallelize the computation of the recurrence (1) is to parallelize the sums directly, using a map-reduce pattern with a *span* [14] of $O(\log k)$ per iteration and $O(k \log k)$ overall after k iterations. This method is attractively simple, but we note that parallelization would provide benefit only once a single iteration provides enough computational work to justify incurring parallelization overheads. In order to extract coarser blocks of parallel work as well as to better control data locality, a *tiling* strategy is developed through construction of a *stencil*. Indeed, experimental results in [19] demonstrate that for a "dense" order 32 ODE ("Exponential Family") at only $k = 64$, the tiling strategy achieves 3x speedup on a 4-core machine. It is not clear if even the 64th iteration of (1) for that ODE would provide enough work to justify parallelization.

The tiling method of [19] proceeds by first tabulating the necessary inner sums of (1) into a table g. Then a triangular system is constructed, shown in (2).

$$\begin{aligned} w_0 \cdot c_d &= b_0 + \textstyle\sum_{i=0}^{d-1} g_{0,i} \cdot c_i \\ w_1 \cdot c_{d+1} &= b_1 + \textstyle\sum_{i=0}^{d-1} g_{1,i} \cdot c_i + g_{1,d} \cdot c_d \\ &\vdots \\ w_k \cdot c_{d+k} &= b_k + \textstyle\sum_{i=0}^{d-1} g_{k,i} \cdot c_i + g_{k,d} \cdot c_d + \cdots + g_{k,d+k-1} \cdot c_{d+k-1}. \end{aligned} \tag{2}$$

The system (2) is then solved by tabulating partial sums in a table T, with entries in T defined by the *stencil* (3,4).

1. For $j \leq i$, we compute

$$T(i,j) := T(i,j-1) + g_{i,d+j-1} \cdot c_{d+j-1}. \tag{3}$$

2. For $i = j$, we additionally compute

$$\begin{aligned} T(i,j) &:= T(i,j)/w_i \\ c_{d+i} &:= T(i,j). \end{aligned} \tag{4}$$

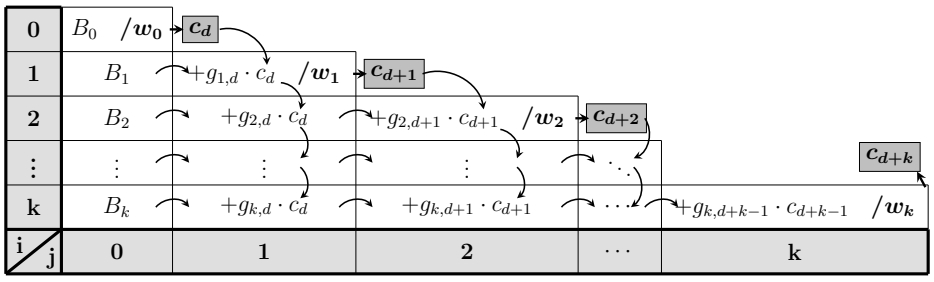

Fig. (2) Constructing the T Table via stencil.

Figure 2 depicts the T table construction, using the notation $B_n = b_n + \sum_{i=0}^{d-1} g_{n,i} \cdot c_i$. This construction creates dependences for a cell in T on cells to its left in the same row and also corner cells (c_{d+j}'s) above. These dependences lead to a recursive division (tiling) of the triangular T table into a rectangular section which recursively divides into four rectangular sections, and two triangular sections which recursively divide further in the same way, shown in Figure 3. This

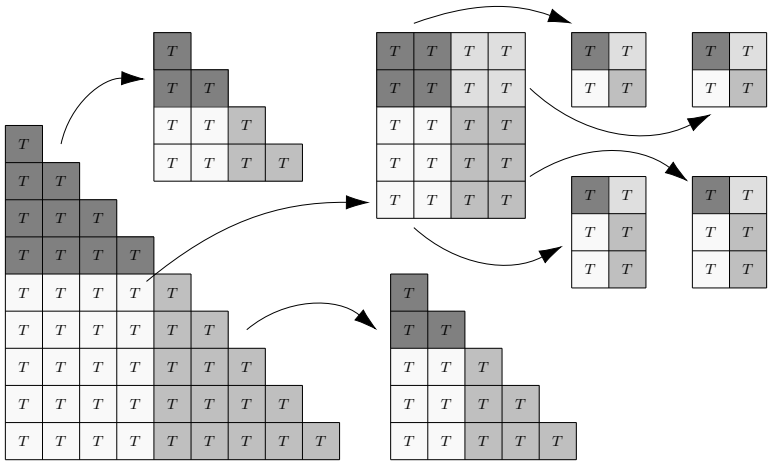

Fig. (3) Recursive tiling of the T Table.

tiling provides better memory locality for accesses to both the g and T tables. It also makes it possible to tune base case sizes for the serial and parallel algorithms to account for coefficient bit-size increases, allowing better cache use and parallel work distribution.

The parallelization of the T table construction follows from noting that dependences allow no parallel recursive calls in a triangular section, but in a rectangular section two recursive calls on the same (left/right) side can be done in parallel. This leads to an algorithm with span $O(k \log k)$. The g table construction has no dependences and can hence be parallelized with span $O(d)$. The *relaxed* nature of the parallelization follows from a small modification described in [19] which allows updates (not starting computations from c_d).

A further theoretical benefit of this tiling strategy is revealed by carefully examining the operations done in a rectangular section. In those sections, at the kth row, the stencil essentially computes $B_k + (g_{k,d}, \ldots, g_{k,d+k-1})(c_d, \ldots, c_{d+k-1})^T$, that is, the dot product of the c_{d+j}'s from corners above with the $g_{k,d+j}$'s. Computing one (or many) independent dot products of size k can be parallelized with total span $O(\log k)$. Therefore the parallel tiling algorithm of [19] could be modified to have have its span improved from $O(k \log k)$ to $O(k)$ - that is, the span would satisfy the recurrence $T(k) = 2T(k/2) + \log k$.

The use of the original tiling algorithm is justified by experimental results presented in [19]. Experiments run on three separate multi-core computers (with cores supporting SMT) demonstrate linear and even super-linear parallel speedup for certain ODEs at large enough expansion order k.

5 Extended Hensel construction via dynamic programming

The extended Hensel construction (EHC) is an algorithm, proposed by Sasaki and Kako in [18], for factoring univariate polynomials over fields of multivariate Puiseux series. For simplicity's sake, we consider here the following restricted situation. Let $F(X, Y) \in \mathbb{C}[X, Y]$, monic and square-free w.r.t. X, with $d := \deg(F, X)$, to be factored into linear factors in X over the field $\mathbb{C}(\langle Y^* \rangle)$ of univariate Puiseux series with coefficients in he field $\mathbb{C}$ of complex numbers:

$$F(X, Y) = (X - \chi_1(Y))(X - \chi_2(Y)) \cdots (X - \chi_d(Y)).$$

The "south-west-most" terms $c\, X^{e_X} Y^{e_Y}$ of $F(X, Y)$, with $c \in \mathbb{C}$, satisfy an equation of the form $e_X/d + e_Y/\delta = 1$, with $\delta \in \mathbb{Q}$ and form the so-called *Newton polynomial* $F^{(0)}(X, Y)$ which is homogeneous $(X, Y^{\delta/d})$ of degree d. Let $\hat{\delta}$ and $\hat{d} \in \mathbb{Z}^{>0}$ be such that: $\hat{\delta}/\hat{d} = \delta/d$, $\gcd(\hat{\delta}, \hat{d}) = 1$. Now we write

$$F(X, Y) \;=\; G_1^{(0)}(X, Y) \cdots G_r^{(0)}(X, Y),$$

where the $G_i^{(0)}(X,Y) := (X - \zeta_i Y^{\delta/d})^{m_i}$ are the so-called *initial factors*. Next, we define the ideal

$$S_k = \langle X^d Y^{(k+0)/\hat{d}}, X^{d-1} Y^{(k+\hat{\delta})/\hat{d}}, \ldots, X^0 Y^{(k+d\hat{\delta})/\hat{d}} \rangle, \tag{5}$$

for $k = 1, 2, \ldots$. Then, for all integers $k > 0$, we can construct $G_i^{(k)}(X,Y) \in \mathbb{C}\langle Y^{1/\hat{d}} \rangle[X]$, for $i = 1, \ldots, r$, satisfying

$$F(X,Y) = G_1^{(k)}(X,Y) \cdots G_r^{(k)}(X,Y) \mod S_{k+1}, \tag{6}$$

and $G_i^{(k)}(X,Y) \equiv G_i^{(0)}(X,Y) \mod S_1$, for all $i = 1, \ldots, r$.

The proof is constructive and by induction on k. We sketch the part of it which yields the algorithmic pattern in which we are interested. Since $F(X,Y) \equiv F^{(0)}(X,Y) \mod S_1$, the theorem is valid for $k = 0$. Let the theorem be valid up to the $(k-1)$-th construction. We write:

$$G_i^{(k-1)} = G_i^{(0)}(X,Y) + \Delta G_i^{(1)}(X,Y) + \cdots + \Delta G_i^{(k-1)}(X,Y),$$

such that $\Delta G_i^{(k')}(X,Y) \in S_{k'}$, and $\deg_X(\Delta G_i^{(k')}(X,Y)) < \deg_X(G_i^{(0)}(X,Y)) = m_i$ for $k' = 1, \ldots, k-1$. These latter properties are part of the induction hypothesis. Now define

$$\Delta F^{(k)}(X,Y) := F(X,Y) - G_1^{(k-1)} \cdots G_r^{(k-1)} \mod S_{k+1}.$$

It follows from the induction hypotheses that $\Delta F^{(k)}(X,Y) \in S_k$ holds. Thus, we can write

$$\Delta F^{(k)}(X,Y) = f_{d-1}^{(k)} X^{d-1} Y^{\hat{\delta}/\hat{d}} + \cdots + f_0^{(k)} X^0 Y^{d\hat{\delta}/\hat{d}}, \tag{7}$$

where $f_\ell^{(k)} = c_\ell^{(k)} Y^{k/\hat{d}}$ and $c_\ell^{(k)} \in \mathbb{C}$ for $\ell = 0, \ldots, d-1$. We construct $G_i^{(k)}(X,Y)$, and thus $\Delta G_1^{(k)}, \ldots, \Delta G_r^{(k)}$, such that we have: $\Delta G_i^{(k)}(X,Y) \equiv 0 \mod S_k$. It turns out that determining the coefficients $c_\ell^{(k)}$ yields the *correction terms* $\Delta G_i^{(k)}(X,Y)$, see Sections 3 and 4 of [1] for details.

Experimentally, the main cost comes from the computation of the *error terms* $\Delta F^{(k)}(X,Y)$. One of the main results of [1] is that these latter terms enjoy relations yielding a *dynamic programming* scheme, which substantially reduce computation costs w.r.t. a direct approach. This dynamic programming scheme is actually an enhanced and generalized version of that of Bernardin in [3], and could hence be parallelized using the ideas therein. Moreover, while the scheme of [1] has not been ported yet to parallel execution, the authors of [1] note that its benefits are demonstrated both practically (via experimentation) and theoretically (via complexity analysis).

6 Concluding remarks

This article has presented several complex algorithmic patterns which have proved effective at parallelizing the computation of series solutions to certain algebraic and differential equations. Lazy evaluation for UPoPS factoring via WPT is parallelized by composing *stencil* and *map-reduce* schemes. Hensel lifting factorization is parallelized by a producer-consumer *pipeline* composed with schemes for WPT. Experiments for these WPT and Hensel lifting schemes demonstrate nearly linear speedup factors on a 12-core node. An EHC based factorization of UPoPS over Puiseux series is made more efficient via a *dynamic programming* "recycling" strategy. This strategy could be parallelized using the ideas of [3]. For solving linear ODEs, a parallel scheme is generated by constructing a table via *stencil* and *recursive tiling*, with experiments achieving linear and even super-linear speedup on multi-core nodes with SMT.

The article has aimed to present these parallel schemes with sufficient detail for their respective problems involving power series, thereby exposing the ideas of the articles [1, 6, 7, 19] to the general symbolic and algebraic computation community, and hopefully motivating the application of the highlighted parallel patterns to other problems in computational mathematics.

References

1. Alvandi, P., Ataei, M., Kazemi, M., Moreno Maza, M.: On the extended Hensel construction and its application to the computation of real limit points. J. Symb. Comput. **98**, 120–162 (2020)
2. Asadi, M., Brandt, A., Chen, C., Covanov, S., Mansouri, F., Mohajerani, D., Moir, R.H.C., Moreno Maza, M., Talaashrafi, D., Solis-Reyes, G.A., Wang, L., Xie, N., Xie, Y.: Basic Polynomial Algebra Subprograms (BPAS) (2021), `www.bpaslib.org`
3. Bernardin, L.: On bivariate hensel and its parallelization. In: ISSAC '98. p. 96–100. Association for Computing Machinery (1998)
4. Blelloch, G.E.: Prefix sums and their applications. CMU-CS ; 90-190, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pa (1990)
5. Bostan, A., Chyzak, F., Ollivier, F., Salvy, B., Schost, É., Sedoglavic, A.: Fast computation of power series solutions of systems of differential equations. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1012–1021. SIAM (2007)
6. Brandt, A., Kazemi, M., Moreno Maza, M.: Power series arithmetic with the BPAS library. In: Computer Algebra in Scientific Computing (CASC '20). LNCS, vol. 12291, pp. 108–128. Springer (2020)
7. Brandt, A., Moreno Maza, M.: On the complexity and parallel implementation of Hensel's lemma and Weierstrass preparation. In: Computer Algebra in Scientific Computing (CASC '21). LNCS, vol. 12865, pp. 78–99. Springer (2021)
8. Chen, C., Covanov, S., Mansouri, F., Moreno Maza, M., Xie, N., Xie, Y.: Parallel integer polynomial multiplication. In: 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 72–80 (2016)
9. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra (3. ed.). Cambridge University Press (2013)
10. Gupta, A., Kumar, V.: Scalability of parallel algorithms for matrix multiplication. In: 1993 International Conference on Parallel Processing. IEEE (1993)
11. Haque, S.A., Mansouri, F., Moreno Maza, M.: On the parallelization of subproduct tree techniques targeting many-core architectures. In: Computer Algebra in Scientific Computing. pp. 171–185. Springer International Publishing (2014)
12. van der Hoeven, J.: Relax, but don't be too lazy. J. Symb. Comput. **34**(6) (2002)
13. Leiserson, C.E.: Encyclopedia of Parallel Computing. Springer US (2011)
14. McCool, M., Reinders, J., Robison, A.: Structured parallel programming: patterns for efficient computation. Elsevier (2012)
15. Moreno Maza, M., Postma, E.: Substituting units into multivariate power series. Maple Trans. **2**(1) (2022)
16. Nissim, R., Schwartz, O., Spiizer, Y.: Fault-tolerant parallel integer multiplication. In: Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures. pp. 207–218. ACM (2024)
17. Nowak, K.J.: Some elementary proofs of Puiseux's theorems. Univ. Iagel. Acta Math **38**, 279–282 (2000)
18. Sasaki, T., Kako, F.: Solving multivariate algebraic equation by Hensel construction. Japan Journal of Industrial and Applied Mathematics **16**(2), 257–285 (1999)
19. Solis-Reyes, G.A., Moreno Maza, M., Brandt, A.: Parallel computation of the power series solutions to linear ordinary differential equations. In: Computer Algebra in Scientific Computing. pp. 380–400. Springer Nature Switzerland (2025)
20. Takahashi, D., Sato, M., Boku, T.: An openmp implementation of parallel fft and its performance on ia-64 processors. In: Voss, M.J. (ed.) OpenMP Shared Memory Parallel Programming. pp. 99–108. Springer Berlin Heidelberg (2003)
21. Trochez, J.P.G., Calder, M., Moreno Maza, M., Postma, E., Romero, M., Brandt, A.: The multivariate power series package in maple 2024. Maple Trans. **5**(2) (2025)
22. Tullsen, D.M., Eggers, S.J., Levy, H.M.: Simultaneous multithreading: maximizing on-chip parallelism. In: ISCA, '95: 22nd Annual International Symposium on Computer Architecture. pp. 392–403. ACM, New York, NY, USA (1995)