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

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

[1] Ontario Research Center for Computer Algebra, University of Western Ontario, Canada

[2] Faculty of Computer Science, Dalhousie University, Canada

ICMS 2026

Overview and Goals

- Present fast and parallel algorithms for
  - Factoring (solving) multivariate polynomials (power series) as power series
  - Factoring multivariate polynomials as Puiseaux series
  - Solving a linear ODE with power series coefficients as power series
- Present non-standard parallel patterns discovered
  - Motivate the use of patterns to other applications
- Present experimental results of `C/C++` implementation (`BPAS`) which validate parallel schemes
  - Sequential versions available in MAPLE `MultivariatePowerSeries` package

Zealous, Lazy and Relaxed

Zealous (*truncated*):

- Treat power series as polynomials
- Allows asymptotically fast methods

Lazy:

- **Compute as needed (term by term)**

Relaxed [van der Hoeven, 2002]:

- Compute extra, anticipating more terms
- Algorithms close to or matching asymptotically fast

The BPAS Library

- Website: www.bpaslib.org
- C/C++
- Implements power series and polynomials over $\mathbb{Q}$ and $\mathbb{Z}$
  - Uses GMP [Granlund et al., 2025]
  - Supports lazy computation
- Implements parallelism tools (C++)

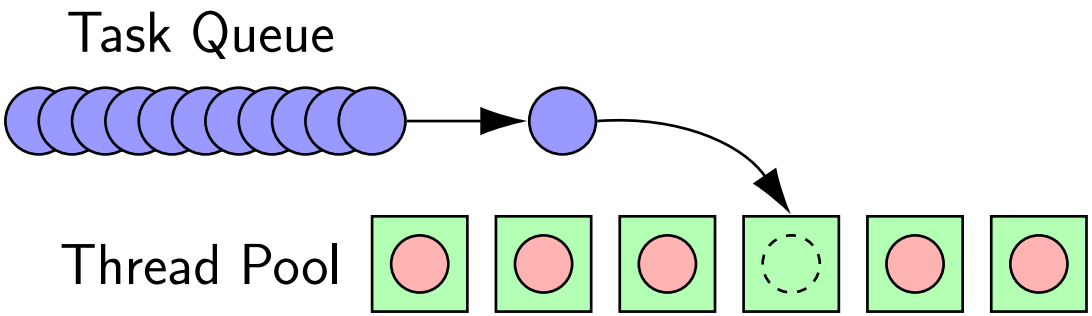


Figure 1: Parallel Thread Pool. Adapted from [Wikipedia, 2025]

The Fork-Join Model

- Suitable for modelling multi-core CPUs
- **Fork**: execute multiple recursive calls in parallel
- **Join**: merge parallel execution back into serial execution
- *"Divide and Conquer"*
  - Divide a problem into sub-problems, solving each recursively
  - Combine sub-solutions to produce a full solution

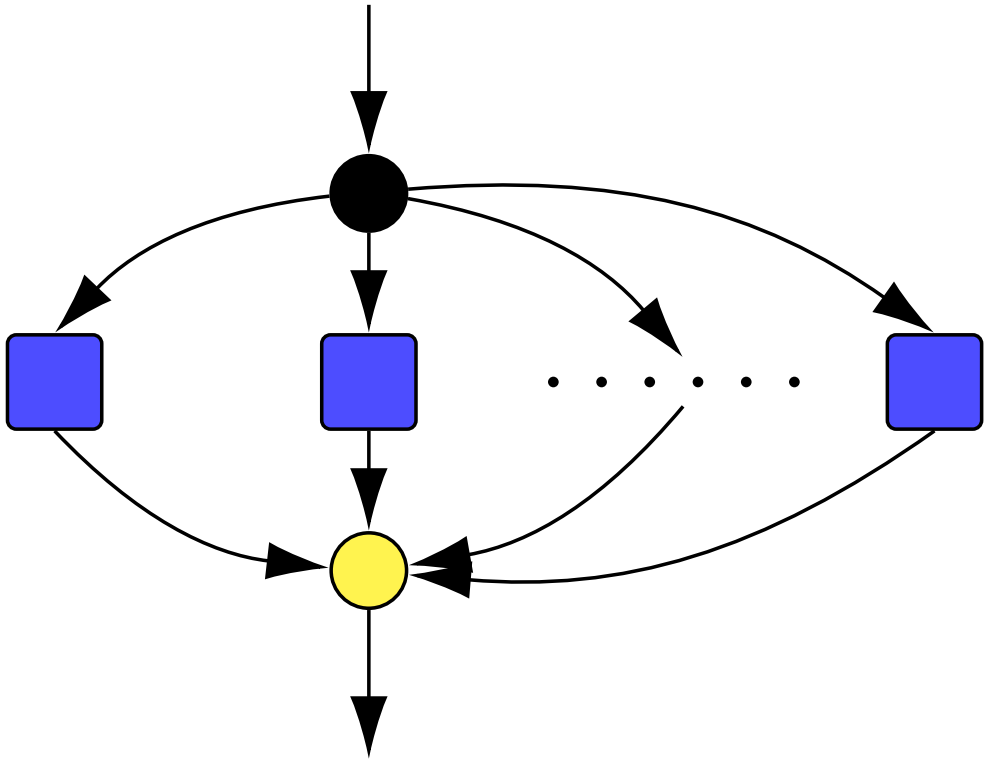


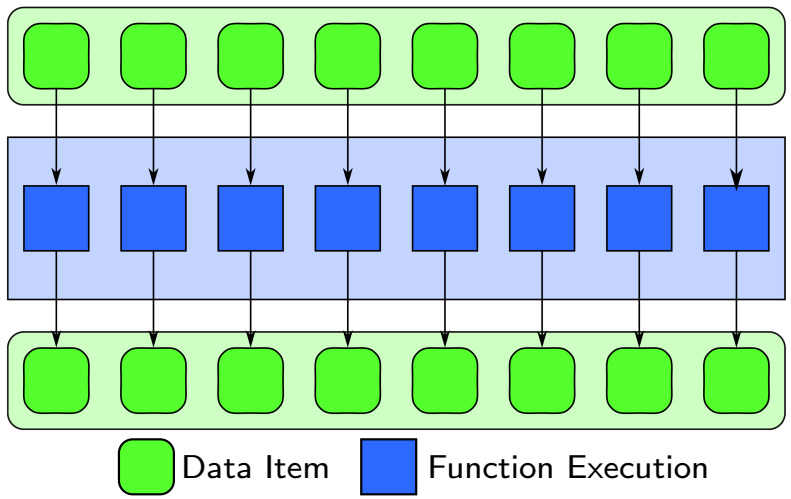
Figure 2: Fork-Join Model. Adapted from [McCool et al., 2012b, Fig. 8.1]

Parallel Map and Workpile

Map - possibly most well-known parallel programming pattern

- execute a function on each item in a collection concurrently
- with multiple Maps, tasks must execute in *lockstep*

Map Pattern [McCool et al., 2012a]



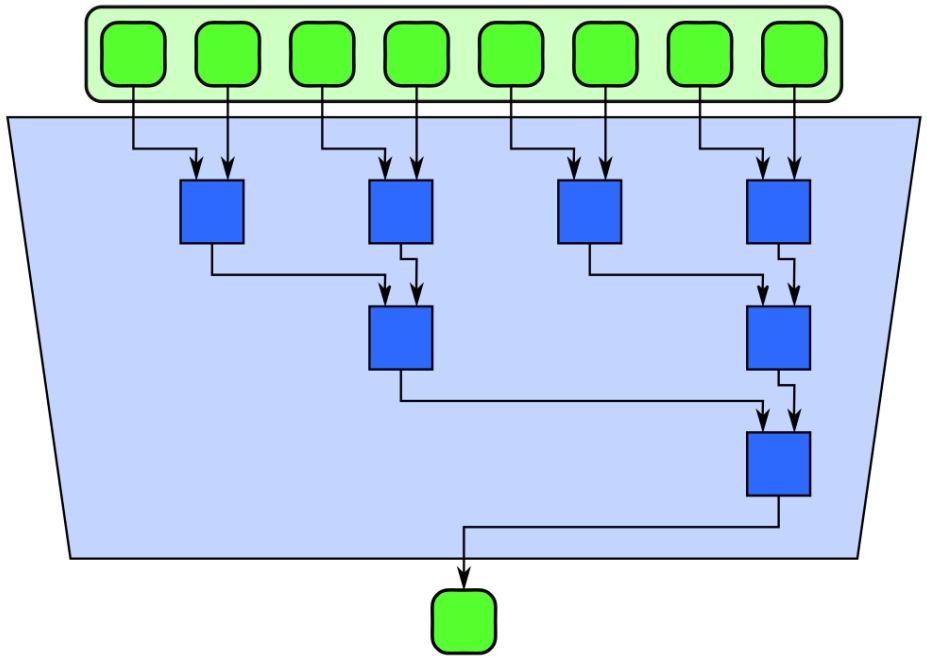
Workpile generalizes Map to a *queue of a tasks*, allowing tasks to add more tasks, thus enabling *load-balancing* as tasks start asynchronously

- one possible implementation of workpile is a **thread pool**

Parallel Reduction

- *Combine* values in parallel

Parallel Reduction [McCool et al., 2012a]



- **Map-Reduce**
  - Map step followed by reduction
  - eg. For loop (non-independent iterations)

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 [Alvandi et al., 2020]
- If f is monic, roots in Y can be computed as multivariate *power series*; we call such a method **Hensel Factorization**

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, $f = g + h$

- $f_{(k)} = g_{(k)} + h_{(k)}$

Multiplication $f = gh$

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

Weierstrass Preparation Theorem for UPoPS

Theorem (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* $b_{d-1}, \ldots, b_0 \in \mathcal{M}$.

Lazy Weierstrass Preparation

Let $f = \sum_{\ell}^{d+m} a_{\ell=0} Y^{\ell}$, $p = Y^d + \sum_{j=0}^{d-1} b_j Y^j$, $\alpha = \sum_{i=0}^{m} c_i Y^i$ be UPoPS. a_ℓ, b_j, c_i are power series. $b_j \in \mathcal{M}$ for $j = 0, \ldots, d-1$

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

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

mod $\mathcal{M}$ we have: (1) $b_j \equiv 0 \mod \mathcal{M}$, $j = 0, \ldots, d-1$

(2) $c_i \equiv a_i \mod \mathcal{M}$ for $i = 0, \ldots, m$

Lazy Weierstrass Phase 1: update p

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

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

- 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

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

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

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

Parallel Challenges

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

- 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} to each thread; number of products computed per thread should be same

Hensel's Lemma

Theorem (Hensel's Lemma)

Let $f = Y^d + \sum_{i=0}^{d-1} a_i Y^i \in \mathbb{K}[[X_1, \ldots, X_n]][Y]$ *be a monic polynomial.* $\bar{f} = f(0, \ldots, 0, Y) = (Y - c_1)^{d_1} \cdots (Y - c_r)^{d_r}$ *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, set $f_r = p(Y - c_r)$, $\widehat{f} = \alpha(Y - c_r)$, and $f = \widehat{f} f_r$. Proceed by induction.

Hensel Factorization

Algorithm 1 HenselFactorization(f)

Require: $f = Y^d + \sum_{i=0}^{d-1} a_i Y^i, a_i \in \mathbb{K}[[X_1, \ldots, X_n]]$.
Ensure: $f_1, \ldots, f_r$ satisfying Hensel's Lemma.

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)
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** $f_1, \ldots, f_r$

Generators and pipelines

Generators

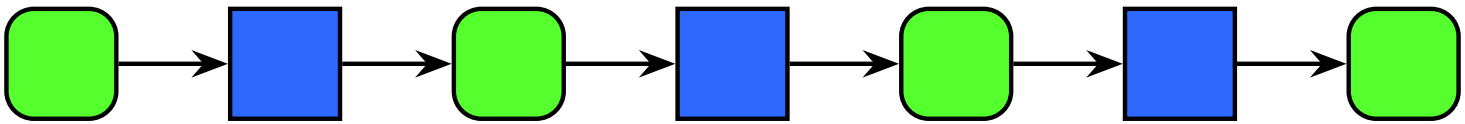
- A generator function (i.e. iterator) yields data items one at a time, allowing the function's control flow to resume on its next execution.

Asynchronous Generators, Producer-Consumer

- *async generators* can concurrently produce items while the generator's caller is consuming items, creating a producer-consumer pair

Pipeline

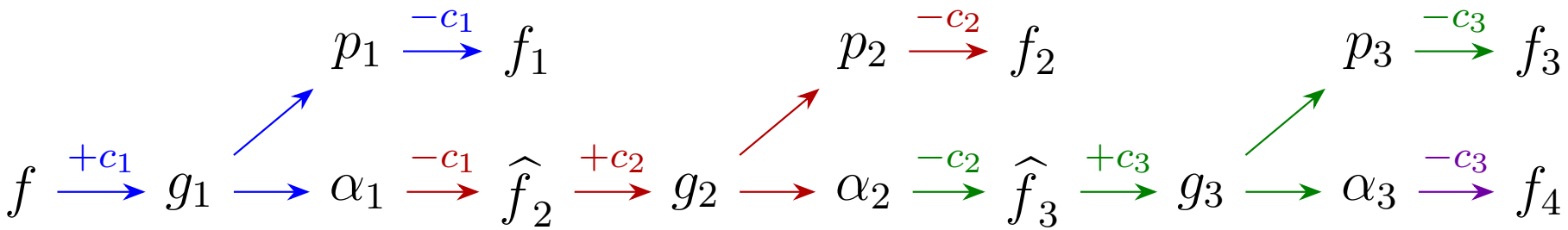
- By connecting many producer-consumer pairs we create a *pipeline*
- Pipelines need not be linear, they can be *directed acyclic graphs*



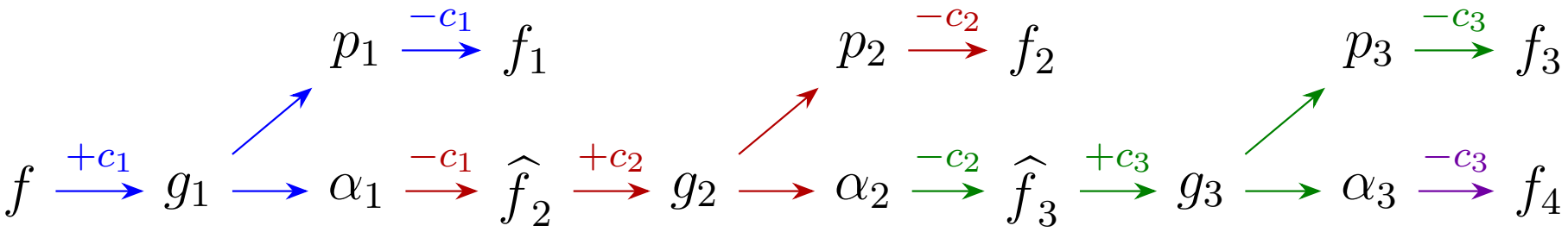
Parallel Opportunities in Hensel

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

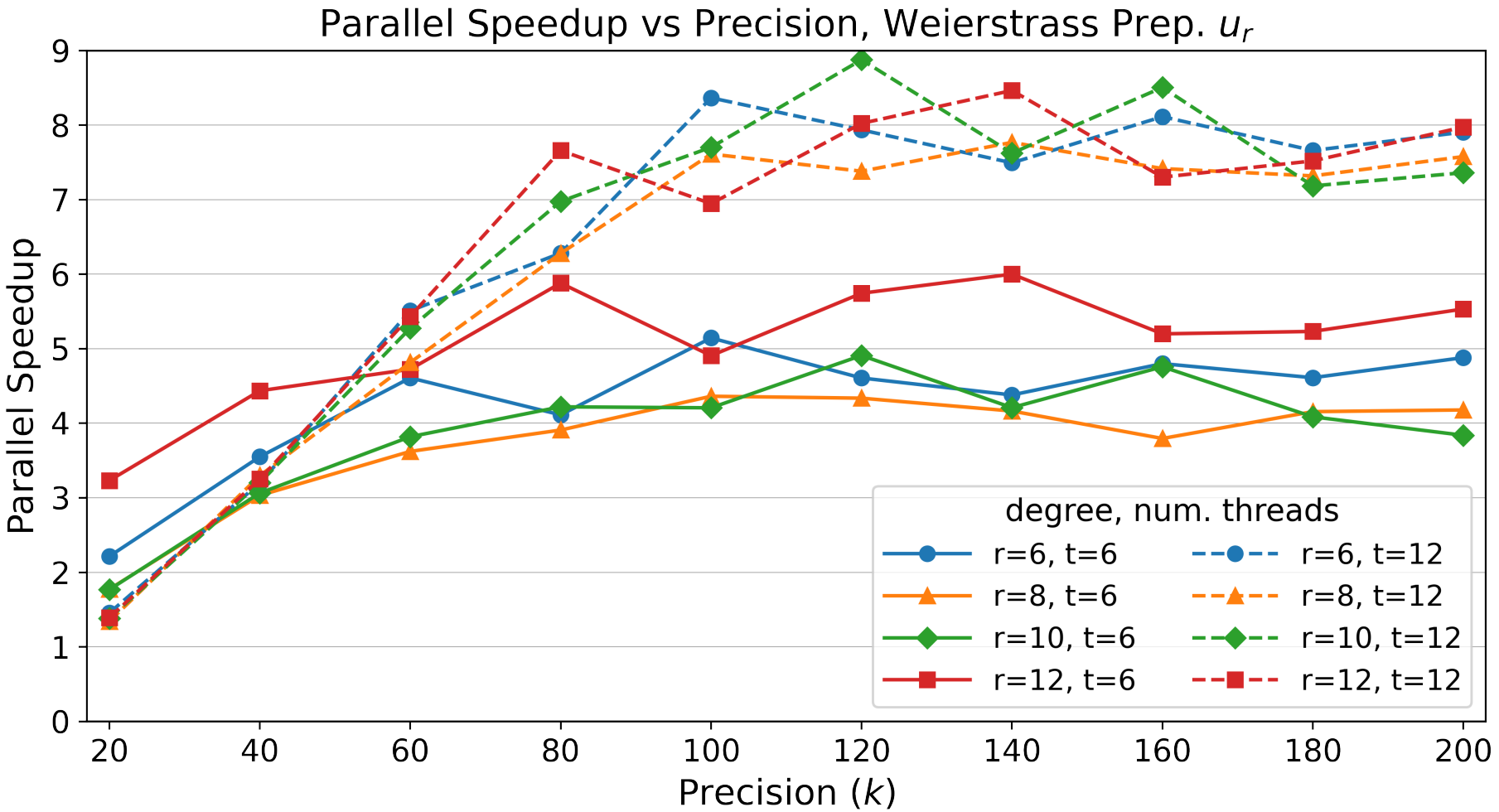


Parallel Challenges and Composition



- 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.

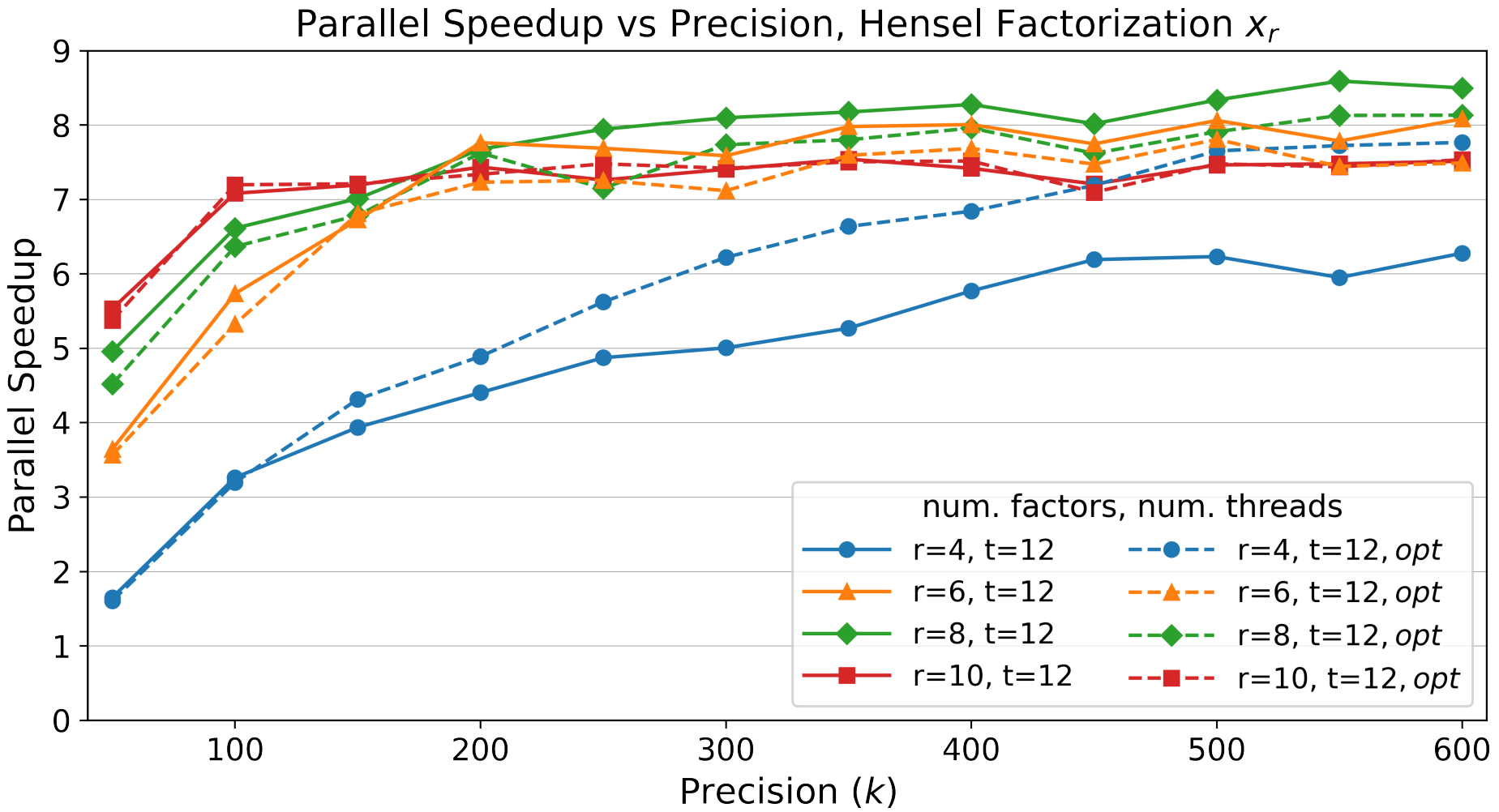
Parallel Speed-up WPT



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

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

Parallel Speed-up Hensel Factorization



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

Problem

Goal

Factorize $F(X,Y) \in \mathbb{C}[X,Y]$ into linear factors in X over $\mathbb{C}(\langle Y^* \rangle)$:

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

where each $\chi_i(Y)$ is a *Puiseux series*.

Puiseux Series

Series of the form

$$\chi_i(Y) = \sum_{k=a_i}^{\infty} c_k Y^{k/d_i}$$

where

- $c_k \in \mathbb{C}$
- $a_i \in \mathbb{Z}$
- $d_i \in \mathbb{Z}_{>0}$

Methods for factoring

1. Extended Hensel Construction (EHC):
   - [Sasaki and Kako, 1999]
   - The Extended Hensel Construction (EHC) computes all branches concurrently
2. Newton-Puiseux:
   - [Kung and Traub, 1978]
   - Approaches based on Newton-Puiseux compute one branch after another

New contributions of the paper [Alvandi et al., 2020]:

- Show that the EHC requires only linear algebra and univariate polynomial arithmetic

Newton Polynomial (1/2)

- Let $F(x,y) \in \mathbb{C}[x,y]$ be square-free, monic in x and let $d := \deg_x(F)$. Also $F(x,0) = x^d$ and F has ≥ 2 terms.
- The "south-west-most" terms $c\,x^{e_x}y^{e_y}$ of $F(x,y)$ satisfy an equation of the form $e_x/d + e_y/\delta = 1$, with $\delta \in \mathbb{Q}$ and form the Newton polynomial $F^{(0)}(x,y)$ which is homogeneous $(x, y^{\delta/d})$ of degree d.
- Let $\widehat{\delta}, \widehat{d} \in \mathbb{Z}^{>0}$ such that: $\widehat{\delta}/\widehat{d} = \delta/d, \quad \gcd(\widehat{\delta}, \widehat{d}) = 1.$

$$F(x,y) \;=\; G_1^{(0)}(x,y)\cdots G_r^{(0)}(x,y)$$

where the $G_i^{(0)}(X,Y) := (X - \zeta_i Y^{\delta/d})^{m_i}$ are the initial factors.

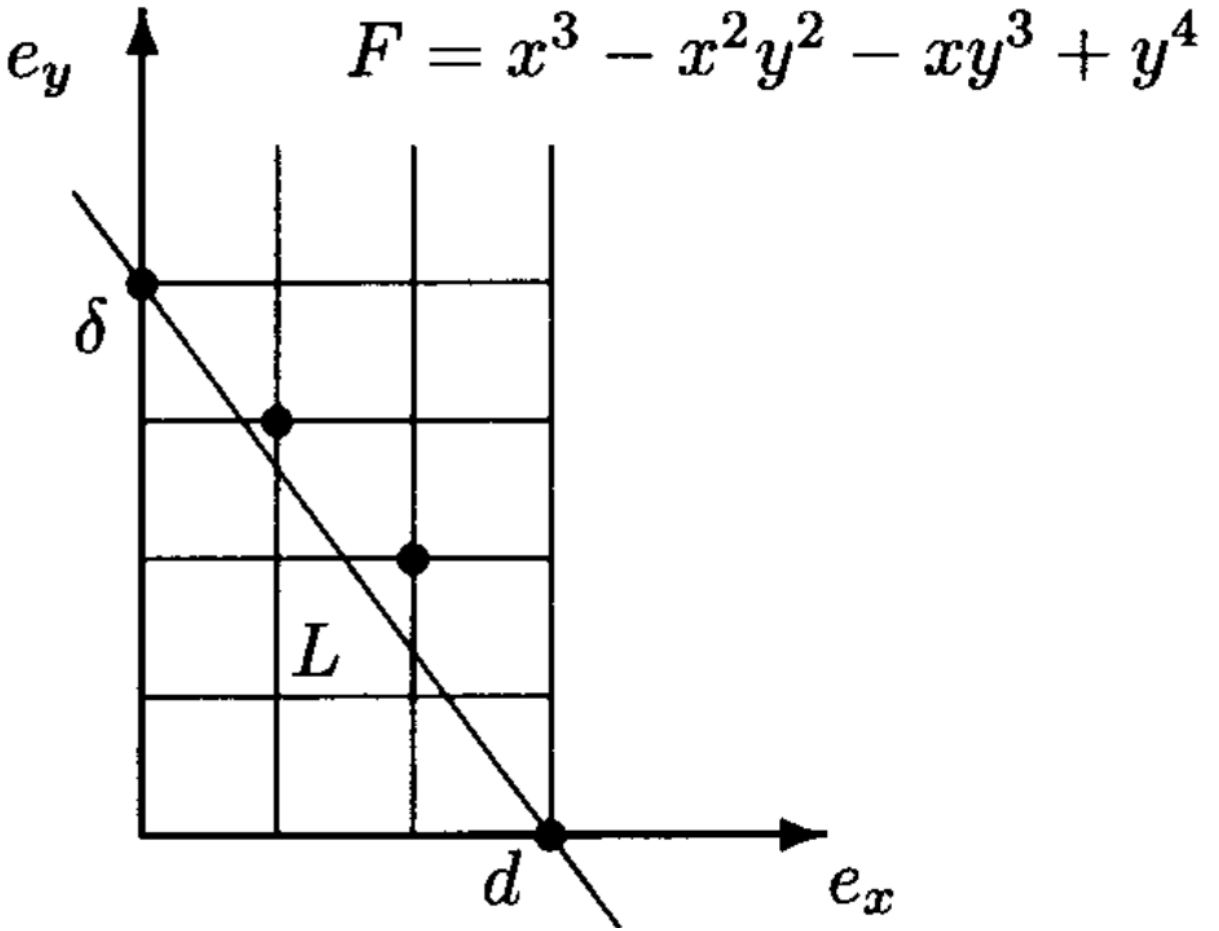
- We define the ideal

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

for $k = 1, 2, \ldots$.

Newton Polynomial (2/2)

Illustration of Newton's line [Kung and Traub, 1978]



Algorithm

Algorithm 2 EHC_Lift(F, k)

1: Compute the Newton polynomial $F^{(0)}$ and $\widehat{\delta}, \widehat{d}$
2: Compute $G_i^{(0)} = (X - \zeta_i Y)^{m_i}$, with $1 \leq i \leq r$
3: Compute the Yun–Moses polynomial $W_i^{(\ell)}$ for $i = 1, \ldots, r$ and $\ell = 0, \ldots, d-1$
4: **for** $j = 1, \ldots, k$ **do**
5: $\quad \Delta F^{(j)}(X, Y) := F(X, Y) - \prod_{i=1}^{r} G_i^{(j-1)} \pmod{\bar{S}_{j+1}}$
6: $\quad \Delta G_i^{(j)} := \sum_{\ell=0}^{m-1} W_i^{(\ell)} f_\ell^{(j)}$, for $i = 1, \ldots, r$
7: $\quad G_i^{(j)} := G_i^{(j-1)} + \Delta G_i^{(j)}$, for $i = 1, \ldots, r$
8: **end for**
9: **return** $G_1^{(k)}, \ldots, G_r^{(k)}$

Yun-Moses Polynomials

Assume $G_1(X,Y),\ldots,G_r(X,Y)$ are homogeneous polynomials. Regarding them as polynomials of $\mathbb{C}\langle Y\rangle[X]$, further assume

$$\gcd((\widehat{G}_i,\widehat{G}_j)) = 1 \text{ for } i \neq j,$$

Let $d := \deg(()G_1(X,Y)\ \ldots\ G_r(X,Y))$. Then, for each $\ell \in \{0,\ldots,d-1\}$, there exists a unique set of polynomials $\{W_i^{(\ell)}(X,Y) \in \mathbb{C}\langle Y\rangle[X] \mid i=1,\ldots,r\}$ satisfying

$$W_1^{(\ell)}\left(\frac{G_1\cdots G_r}{G_1}\right) + \cdots + W_r^{(\ell)}\left(\frac{G_1\cdots G_r}{G_r}\right) = X^\ell Y^{d-\ell},$$

where $\deg_X\left(W_i^{(\ell)}(X,Y)\right) < \deg_X\left(G_i(X,Y)\right), \quad i=1,\ldots,r.$

- We can compute the Yun-Moses Polynomials using linear algebra operations

Computing $\Delta F^{(j)}(X,Y)$

Goal

$$\Delta F^{(j)}(X,Y) := F(X,Y) - \prod_{i=1}^{r} G_i^{(j-1)} \mod \bar{S}_{j+1}$$

Observation

- $G_i^{(j-2)} := G_i^{(0)} + \Delta G_i^{(1)} + \cdots + \Delta G_i^{(j-2)}$
- $G_i^{(j-1)} := G_i^{(0)} + \Delta G_i^{(1)} + \cdots + \Delta G_i^{(j-2)} + \Delta G_i^{(j-1)}$

Hence, we aim at recycling terms in the product $\prod_{i=1}^{r} G_i^{(j-1)} \mod \bar{S}_{j+1}$ computed from previous iterations.

Notations

1. $P_2^{k+1} := \prod_{i=1}^{2} G_i^{(k)} \mod \overline{S}_{k+1}$
2. $P_j^{k+1} := \prod_{i=1}^{j} G_i^{(k)} \mod \overline{S}_{k+1}$, for $j = 3, \ldots, r$.

We want

$$P_r^{k+1} = \prod_{i=1}^{r} G_i^{(k)} \mod \overline{S}_{k+2}$$

Computing $\Delta F^{(j)}(X, Y)$

Initially define: $P_j^1 \equiv G_1^{(0)} \cdots G_j^{(0)} \mod S_2$, for $j = 2, \cdots, r$. and recursively compute:

$$P_2^{k+1} = P_2^k + (\Delta_1^0 \Delta_2^k + \Delta_1^k \Delta_2^0)\tilde{Y}^k + (\Delta_1^1 \Delta_2^k + \cdots + \Delta_1^k \Delta_2^1)\tilde{Y}^{k+1} = \prod_{i=1}^{2} G_i^{(k)}$$

Now for $j = 3, \ldots, r$, define

$$P_j^k \equiv P_{j-1}^k G_j^{(k-1)} \mod S_{k+1}$$

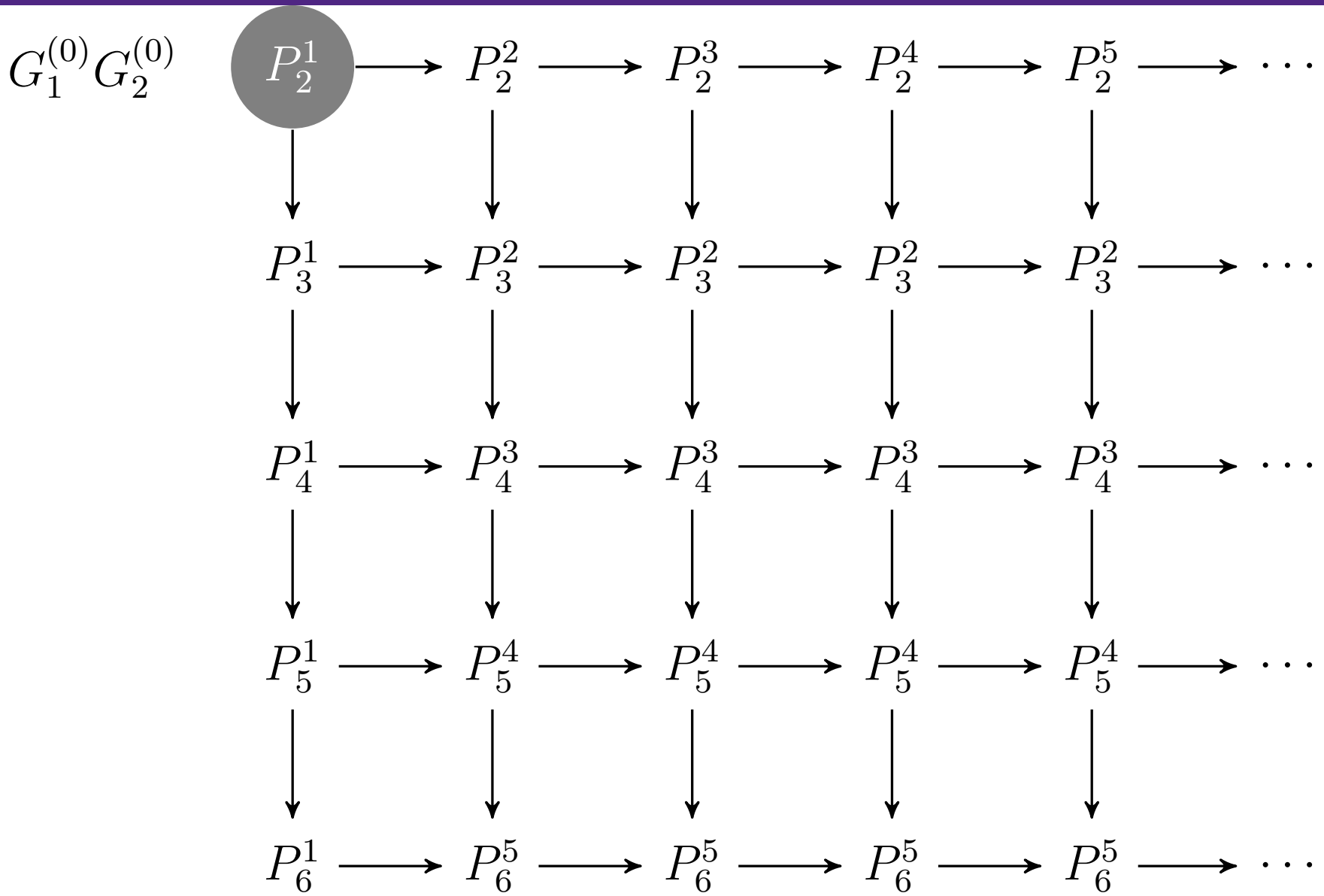
and assume q_j^{k+1} is recursively given by

$$q_j^{k+1} = p_{j-1}^{k+1,0} \Delta_j^k + q_{j-1}^{k+1} \Delta_j^0 \text{ with } q_2^{k+1} = \Delta_2^k \Delta_1^0 + \Delta_2^0 \Delta_1^k.$$

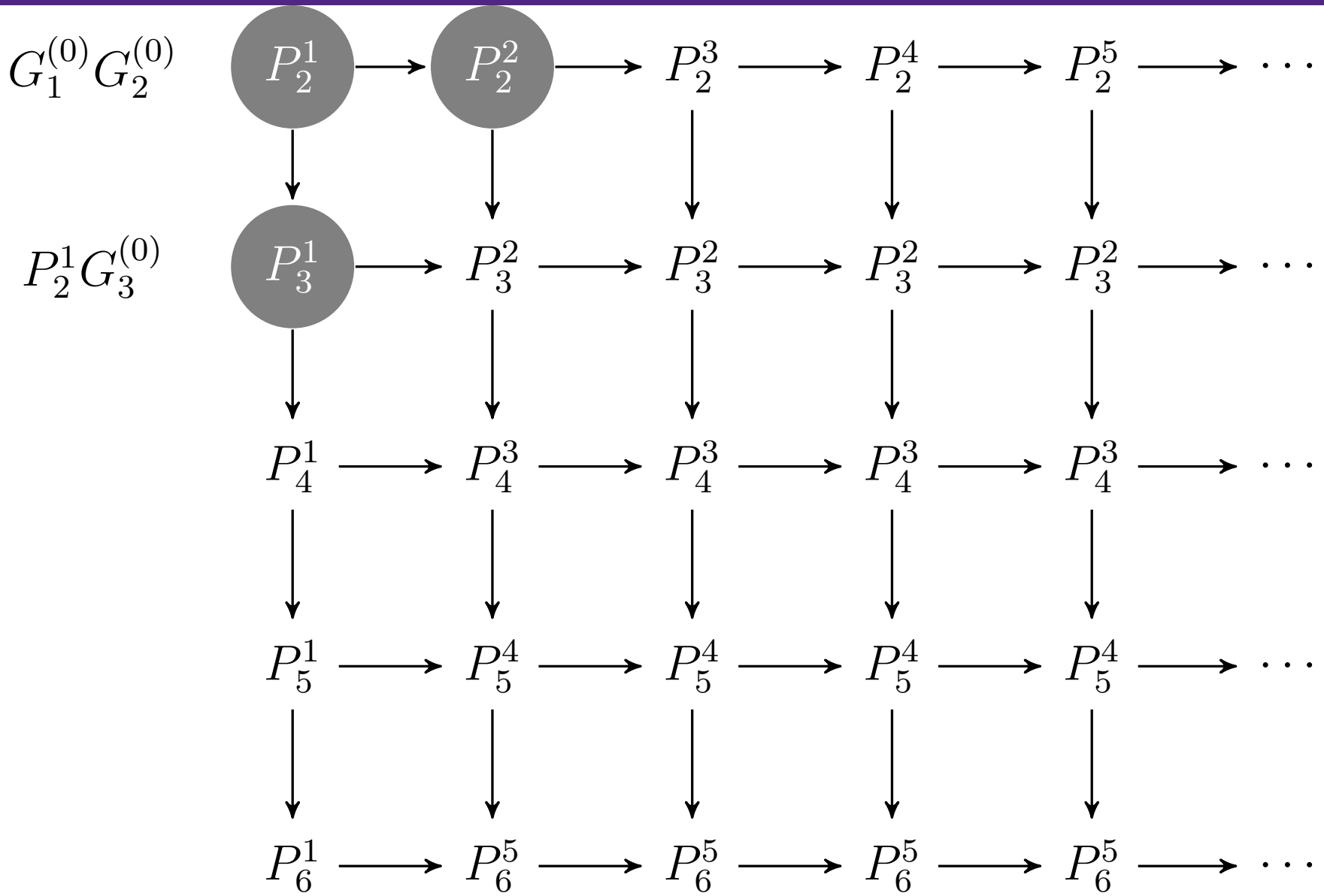
where $p_{j-1}^{k+1,0}$ is the coefficient of $\tilde{Y}^0$ in P_{j-1}^{k+1}. We can compute

$$P_j^{k+1} = P_j^k + q_j^{k+1}\tilde{Y}^k + \left(p_{j-1}^{k+1,1}\Delta_j^k + \cdots + p_{j-1}^{k+1,k+1}\Delta_j^0\right)\tilde{Y}^{k+1} = \prod_{i=1}^{j} G_i^{(k)}$$

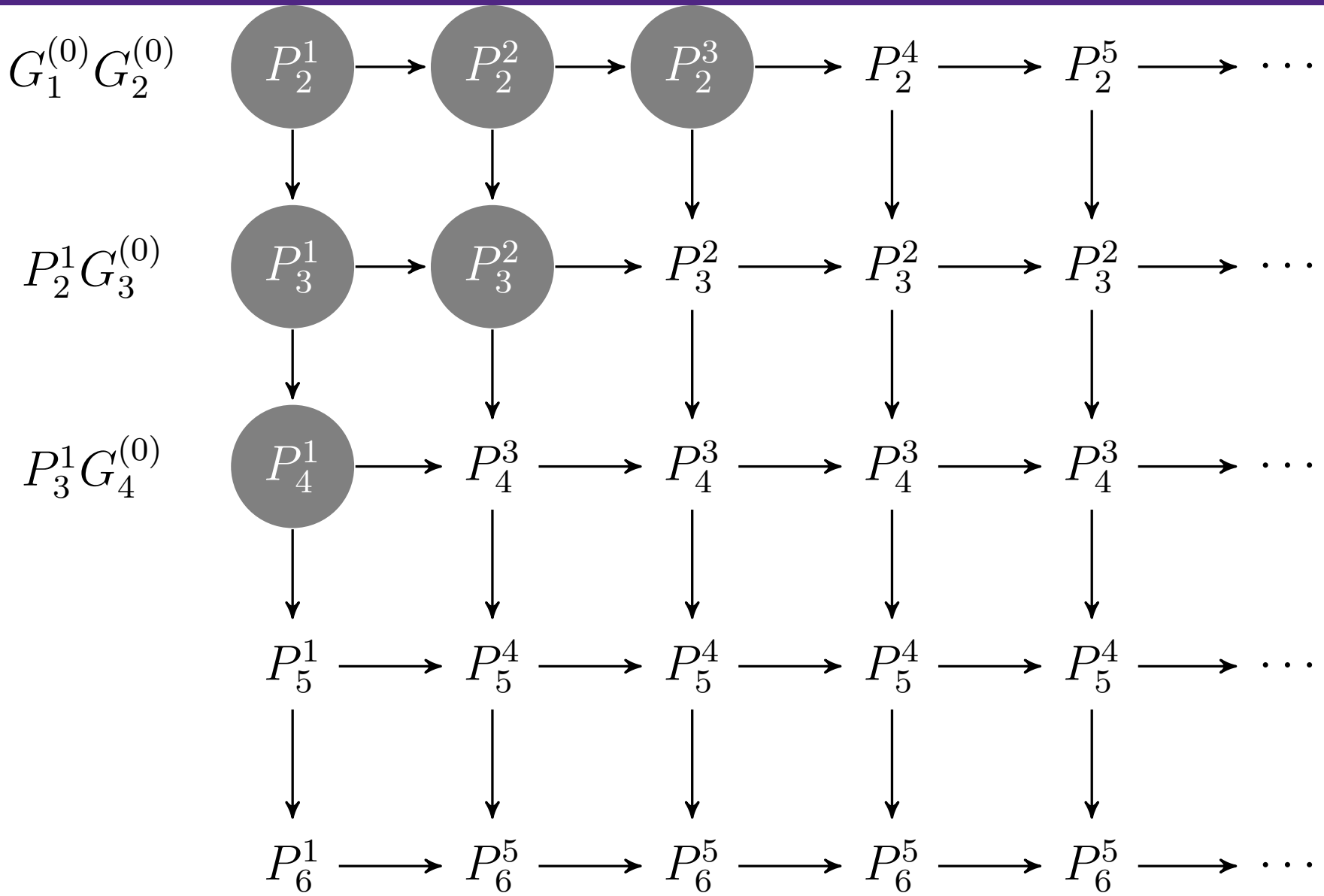
Computing $\Delta F^{(j)}(X, Y)$



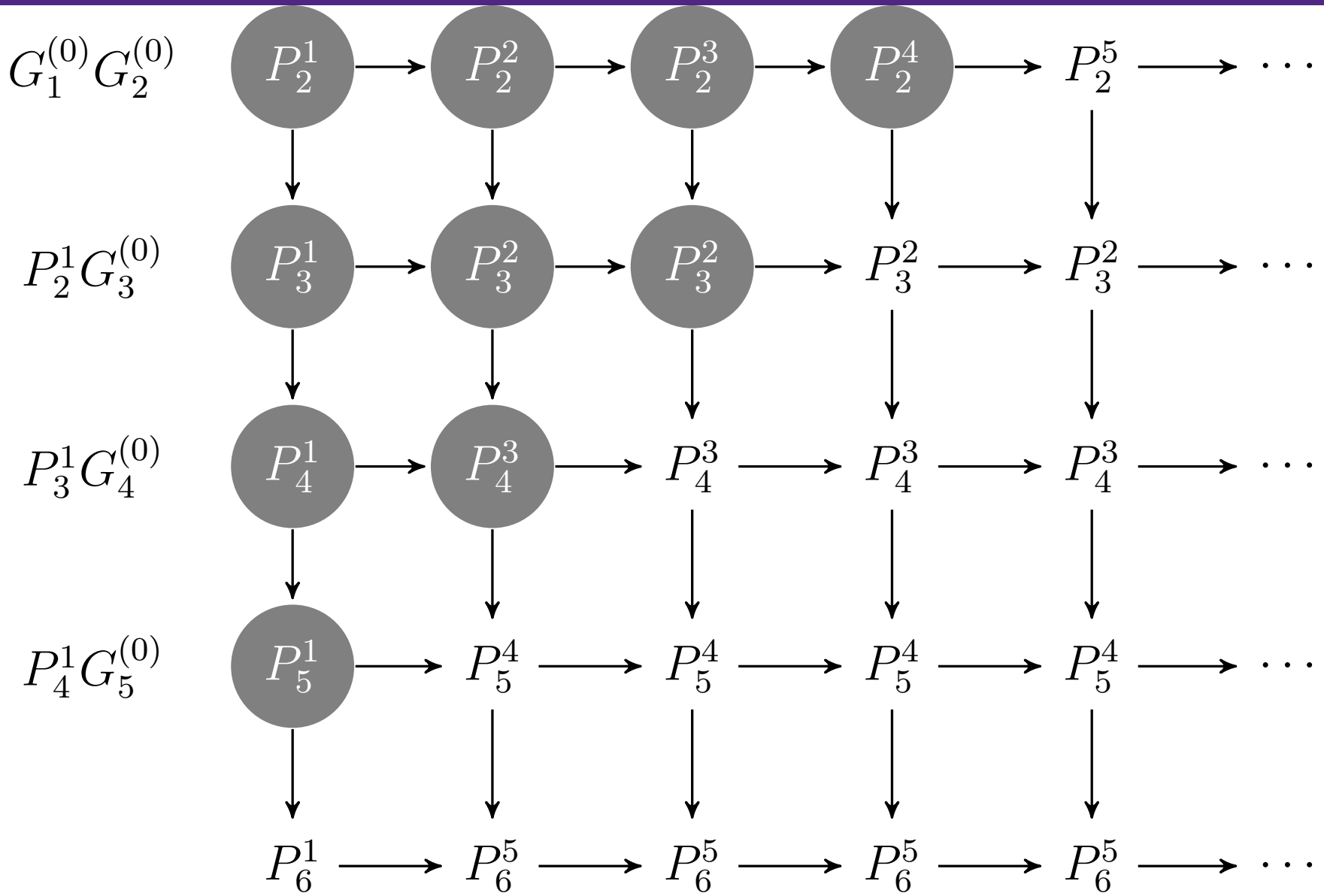
Computing $\Delta F^{(j)}(X, Y)$



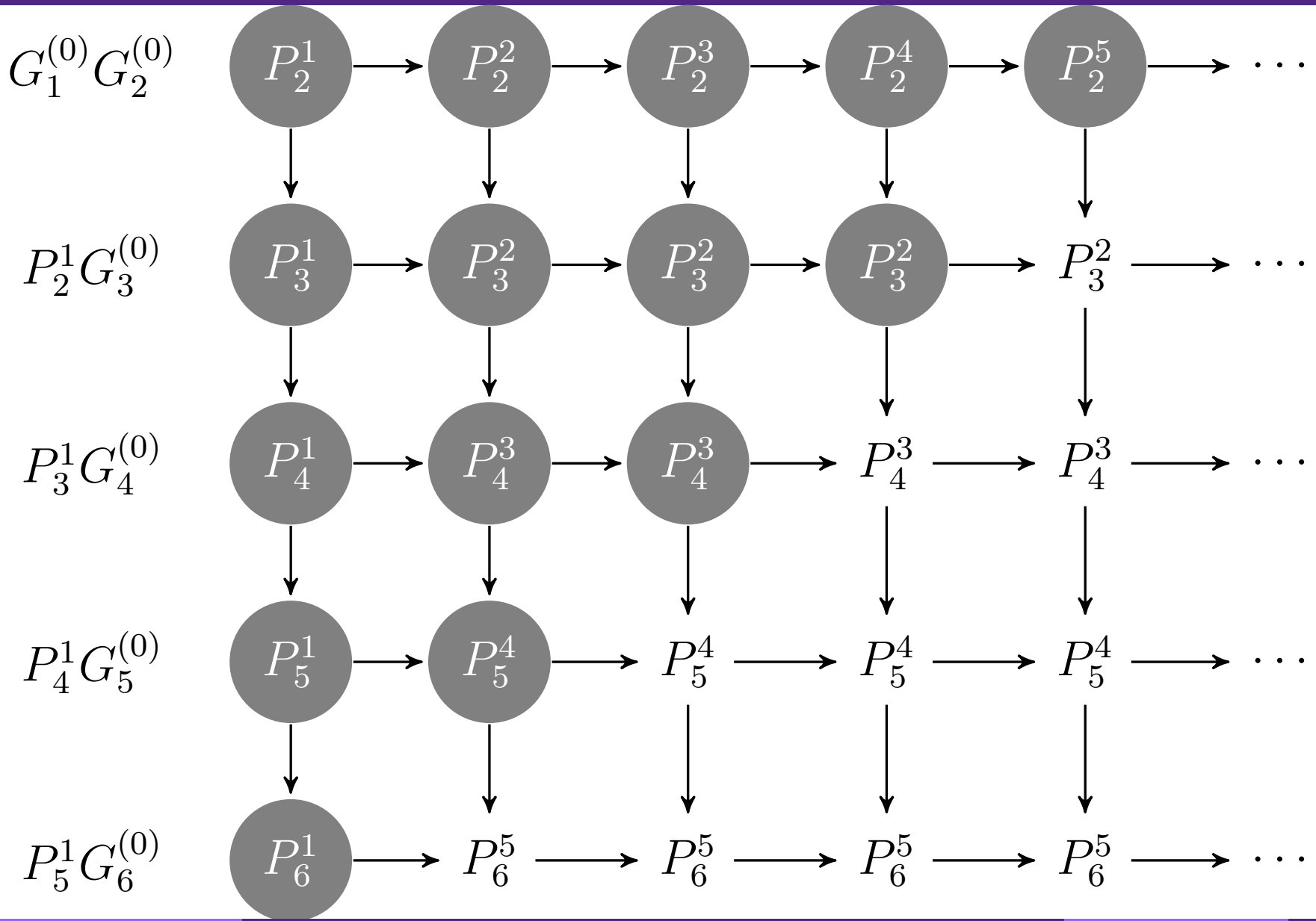
Computing $\Delta F^{(j)}(X, Y)$



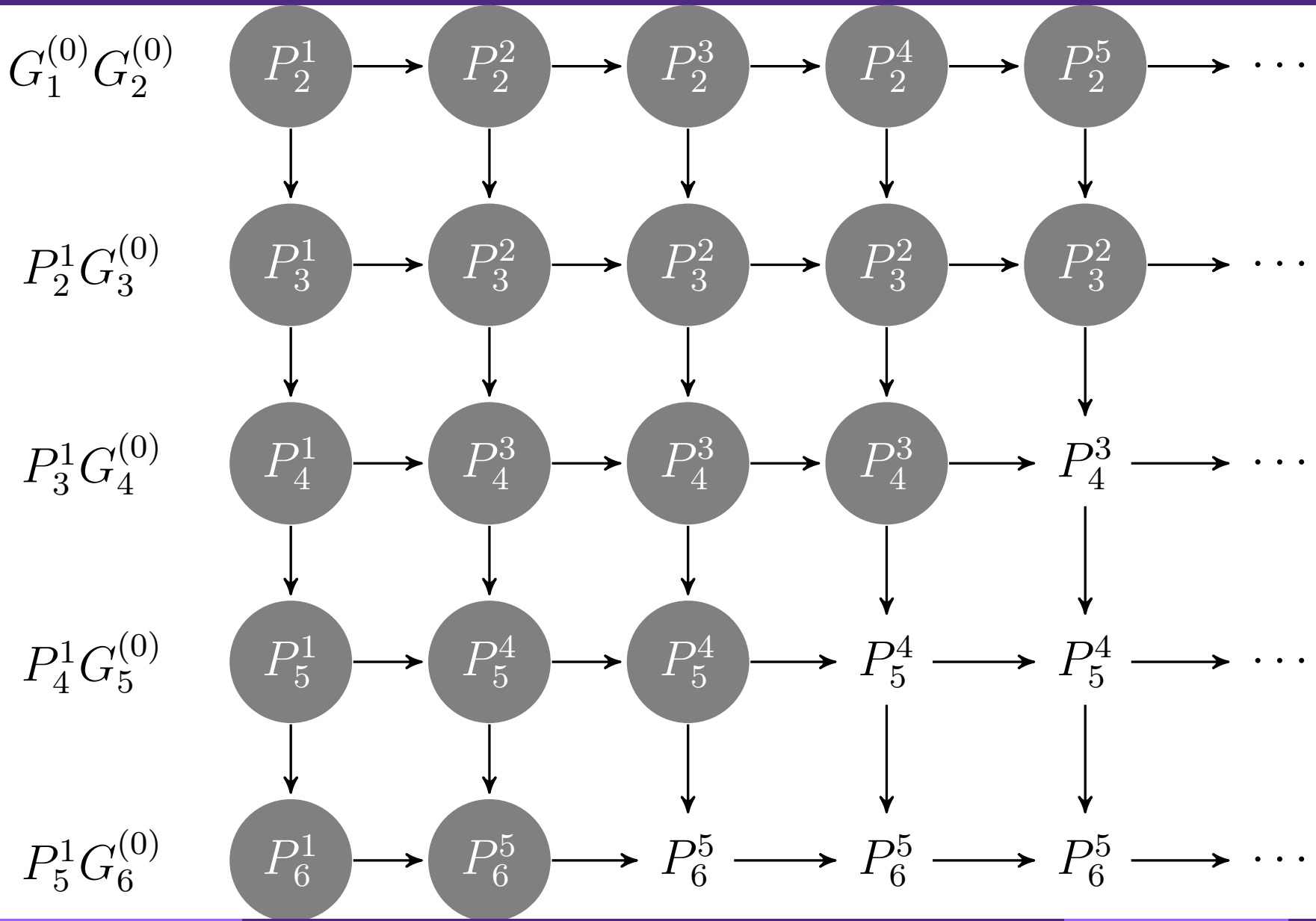
Computing $\Delta F^{(j)}(X, Y)$



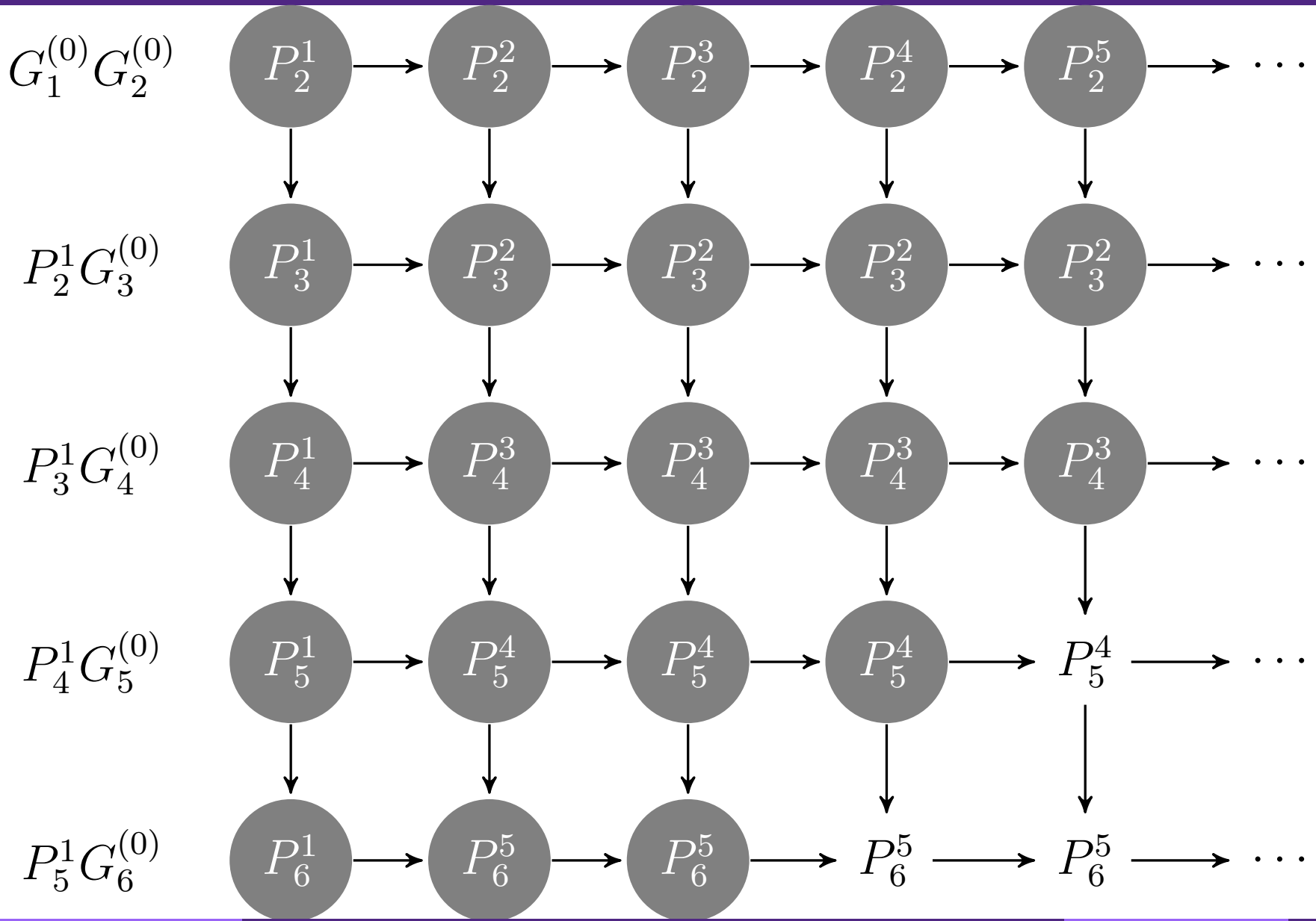
Computing $\Delta F^{(j)}(X, Y)$



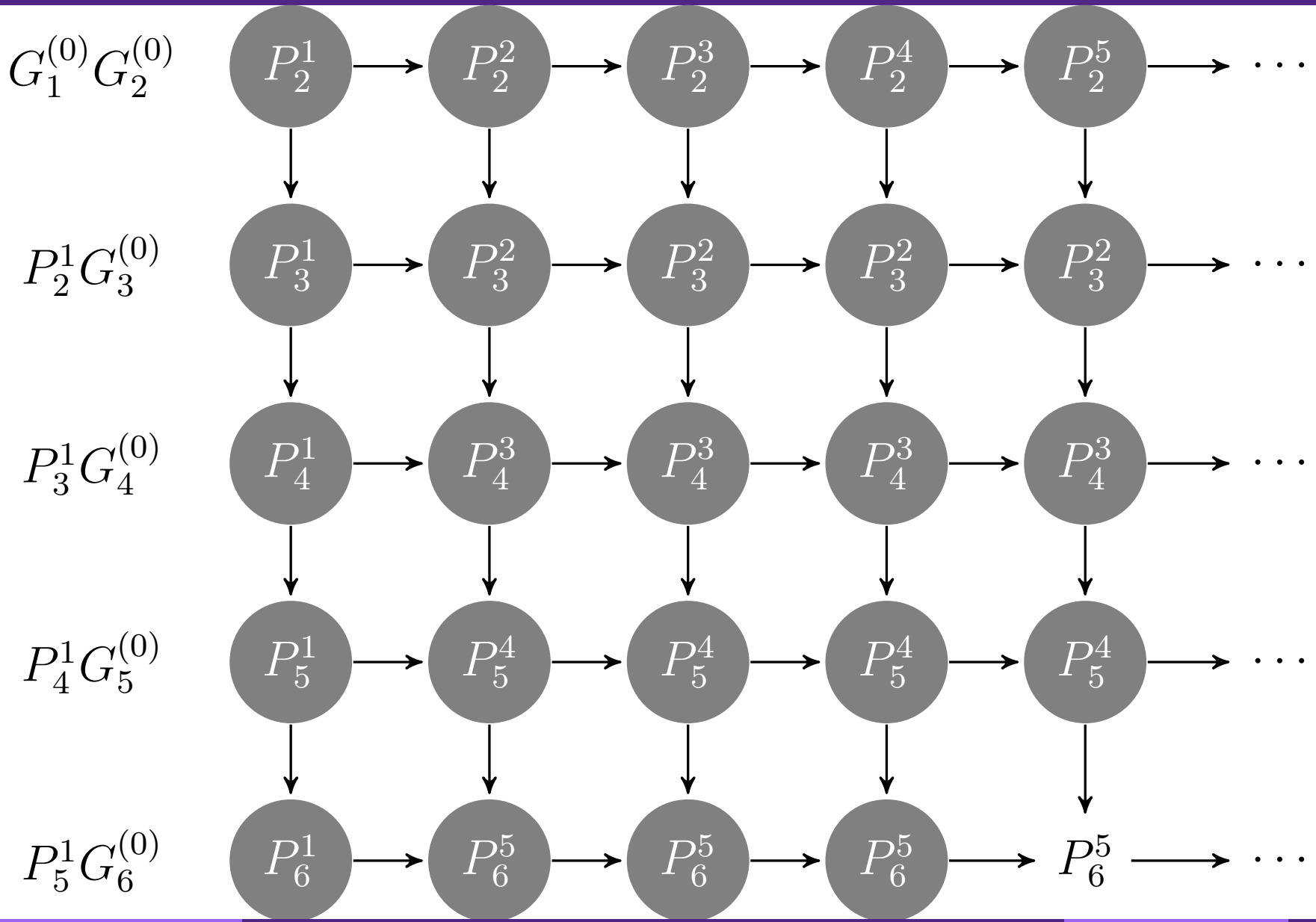
Computing $\Delta F^{(j)}(X, Y)$



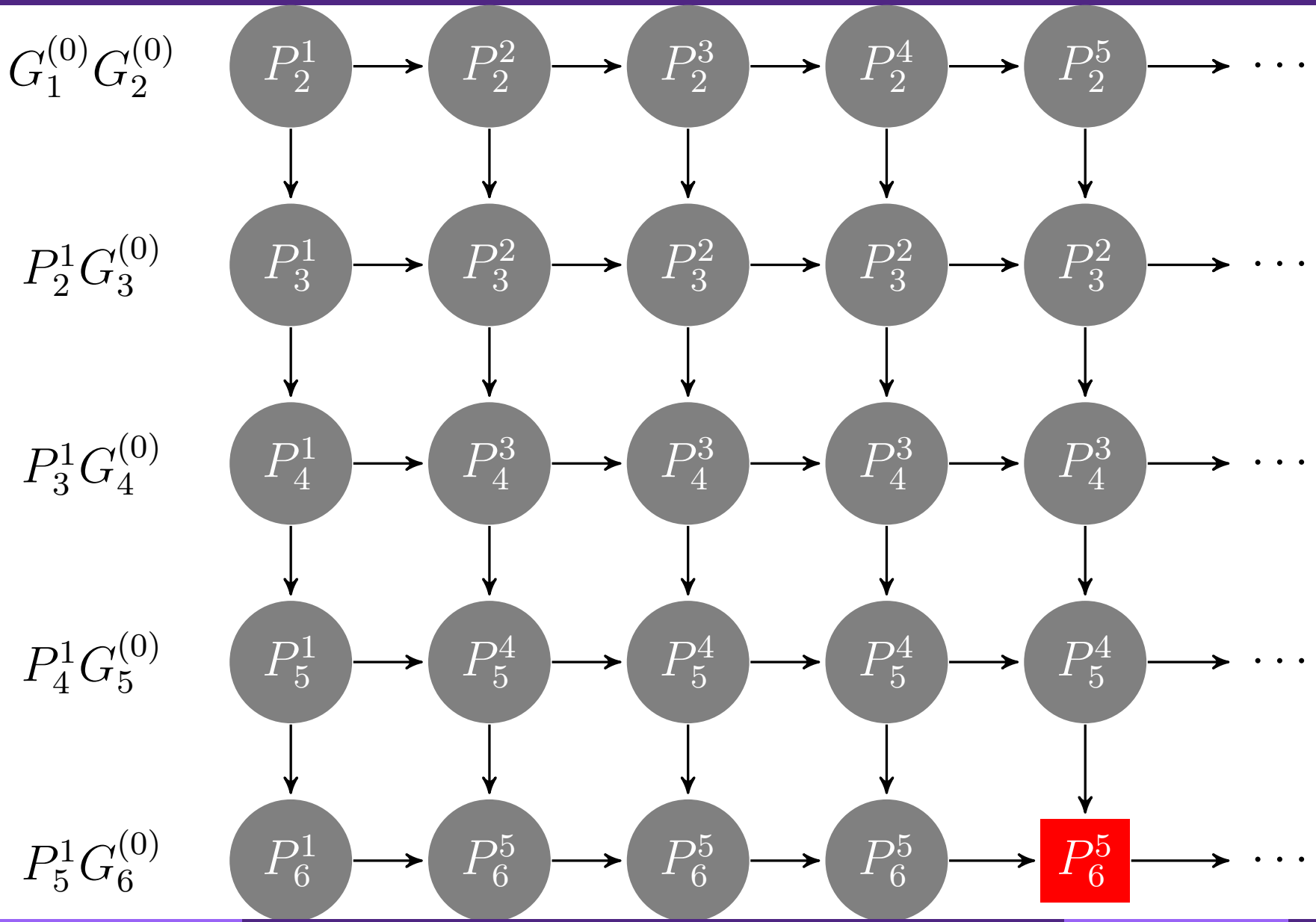
Computing $\Delta F^{(j)}(X, Y)$



Computing $\Delta F^{(j)}(X, Y)$



Computing $\Delta F^{(j)}(X, Y)$



Paralellization

Parallelization follows from:

- Parallel linear algebra for Yun-Moses polynomial computations
- Parallel power series and polynomial operations
- Paralellizing dynamic programming tiling for $\Delta F^{(j)}(X, Y)$ by computing diagonals concurrently

Experimentation for factoring

Ex	MD	KT Lin		KT Quad		EHCWM			EHCEEA	
		KT10	KT20	KT10	KT20	EHC10	YM1	EHC20	EHC10	YM2
17	7	145	∞	23.4	23.2	0.78	0.43	3.37	4.12	1.77
19	4	0.14	0.16	0.16	0.14	0.39	0.26	0.45	0.51	0.15
20	4	2.79	7.98	0.77	0.82	0.26	0.15	0.29	0.50	0.24
21	4	8.58	143	1.96	1.93	0.23	0.12	0.31	0.47	0.16
24	5	2.90	24.8	1.11	1.11	0.26	0.15	0.35	0.49	0.17
25	7	1.83	9.45	0.90	1.00	0.46	0.31	0.50	0.73	0.42
26	8	2.35	12.3	3.09	3.29	0.66	0.53	0.74	2.18	1.80
27	8	60.8	2876	23.1	27.1	0.77	0.53	1.20	2.31	1.28
28	9	215	∞	73.8	123	1.88	1.03	2.11	7.03	4.92
30	17	∞	∞	∞	∞	39.8	6.70	41.3	53.8	16.5
31	32	∞	∞	∞	∞	599	24.9	∞	∞	∞
32	33	∞	∞	∞	∞	224	25.0	∞	∞	∞

Table 1: Comparing EHC versus Kung-Traub's method

Examples from `regularchains.org`

Maple commands for UPoPS factoring

In the MultivariatePowerSeries package

```
> with(MultivariatePowerSeries);
[Add, ApproximatelyEqual, ApproximatelyZero, ChangeOfVariables, Copy, Degree,
    Display, Divide, EvaluateAtOrigin, Exponentiate,
    ExtendedHenselConstruction, GeometricSeries, GetAnalyticExpression,
    GetCoefficient, GetMonomial, GetOrder, GetPowerSeries, GetPowerSeriesOrder,
    GetPuiseuxSeriesOrder, GetRays, HenselFactorize, HomogeneousPart, Inverse,
    IsUnit, MainVariable, Multiply, Negate, PowerSeries, Precision,
    PuiseuxFactorize, PuiseuxSeries, SetDefaultDisplayStyle, SetDisplayStyle,
    SetHenselBound, SetNonzeroPowerSeriesDegreeBound, SetPuiseuxBound,
    SetSmallestTermDegreeBound, Substitute, Subtract, SumOfAllMonomials,
    TaylorShift, Truncate, TschirnhausenTransformation,
    UnivariatePolynomialOverPowerSeries, UnivariatePolynomialOverPuiseuxSeries,
    UpdatePrecision, Variables, WeierstrassPreparation]
>
```

Problem Statement

Compute the first $d + k + 1$ terms of the power series solving a single linear ODE:

$$a_d(x)y^{(d)}(x) + a_{d-1}(x)y^{(d-1)}(x) + \cdots \\ \cdots + a_1(x)y'(x) + a_0(x)y(x) = b(x) \quad (2)$$

- d - order, k - number of new terms
- $a_d(x), a_{d-1}(x), \ldots, a_0(x), b(x)$ are power series ($\in \mathbb{Q}[[x]]$)
- $a_d(x)$ is invertible ($a_d(0) \neq 0$)
- $y(0), y'(0), \ldots, y^{(d-1)}(0)$ are given as initial conditions

Notation

$$a_d(x)y^{(d)}(x) + a_{d-1}(x)y^{(d-1)}(x) + \cdots$$
$$\cdots + a_1(x)y'(x) + a_0(x)y(x) = b(x) \quad (2)$$

$$a_i(x) = \sum_{n=0}^{\infty} a_{i,n}x^n, \quad b(x) = \sum_{n=0}^{\infty} b_n x^n, \quad y(x) = \sum_{n=0}^{\infty} c_n x^n$$

$$y'(x) = \sum_{n=1}^{\infty} n c_n x^{n-1}, \quad y''(x) = \sum_{n=2}^{\infty} n(n-1)c_n x^{n-2}$$

$$y^{(j)}(x) = \sum_{n=j}^{\infty} \frac{n!}{(n-j)!} c_n x^{n-j} = \sum_{n=0}^{\infty} \frac{(n+j)!}{n!} c_{n+j} x^n$$

The Recurrence Equation

$$c_{d+k} = \frac{1}{w_k} \cdot \left(b_k - \sum_{i=0}^{d+k-1} \left(c_i \sum_{m=0}^{d} r_{m,i-m} \cdot a_{m,k-i+m} \right) \right) \tag{3}$$

$$r_{m,i-m} \cdot a_{m,k-i+m} \equiv 0 \quad \textbf{if } i < m \textbf{ or } k < i - m$$

- $c_0, \ldots, c_{d-1}$ from initial conditions
- $w_k = a_{d,0} \cdot r_{d,k}$

We extract the inner sums:

$$g_{n,i} = -\sum_{m=0}^{d} \begin{cases} r_{m,i-m} \cdot a_{m,n-i+m} & \text{if } i \geq m \textbf{ and } n \geq i - m \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Tabulating the Inner Sums

When calculating many c_{d+n}'s (up to c_{d+k}):
$(0 \leq n \leq k, \quad 0 \leq i < d+n)$

$$
\begin{aligned}
w_0 \cdot c_d &= b_0 + \textstyle\sum_{i=0}^{d-1} g_{0,i} \cdot c_i \\
w_1 \cdot c_{d+1} &= b_1 + \textstyle\sum_{i=0}^{d-1} g_{1,i} \cdot c_i + g_{1,d} \cdot c_d \\
w_2 \cdot c_{d+2} &= b_2 + \textstyle\sum_{i=0}^{d-1} g_{2,i} \cdot c_i + g_{2,d} \cdot c_d + g_{2,d+1} \cdot c_{d+1} \\
&\vdots \\
w_k \cdot c_{d+k} &= b_k + \textstyle\sum_{i=0}^{d-1} g_{k,i} \cdot c_i + g_{k,d} \cdot c_d + g_{k,d+1} \cdot c_{d+1} + \cdots \\
&\qquad + g_{k,d+k-1} \cdot c_{d+k-1}
\end{aligned}
\tag{5}
$$

We can tabulate all the g's (g Table):

- $d \cdot (k+1) + \sum_{i=0}^{k} i \;=\; d(k+1) + k(k+1)/2$

Triangular System

Notation for sums from the initial conditions and $b(x)$:

$$B_n = b_n + \sum_{i=0}^{d-1} g_{n,i} \cdot c_i$$

We have a triangular system:

$$\begin{aligned}
w_0 \cdot c_d &= B_0 \\
w_1 \cdot c_{d+1} &= B_1 + g_{1,d} \cdot c_d \\
w_2 \cdot c_{d+2} &= B_2 + g_{2,d} \cdot c_d + g_{2,d+1} \cdot c_{d+1} \\
&\vdots \\
w_k \cdot c_{d+k} &= B_k + g_{k,d} \cdot c_d + g_{k,d+1} \cdot c_{d+1} + \cdots \\
&\qquad + g_{k,d+k-1} \cdot c_{d+k-1}
\end{aligned} \tag{6}$$

The T Table

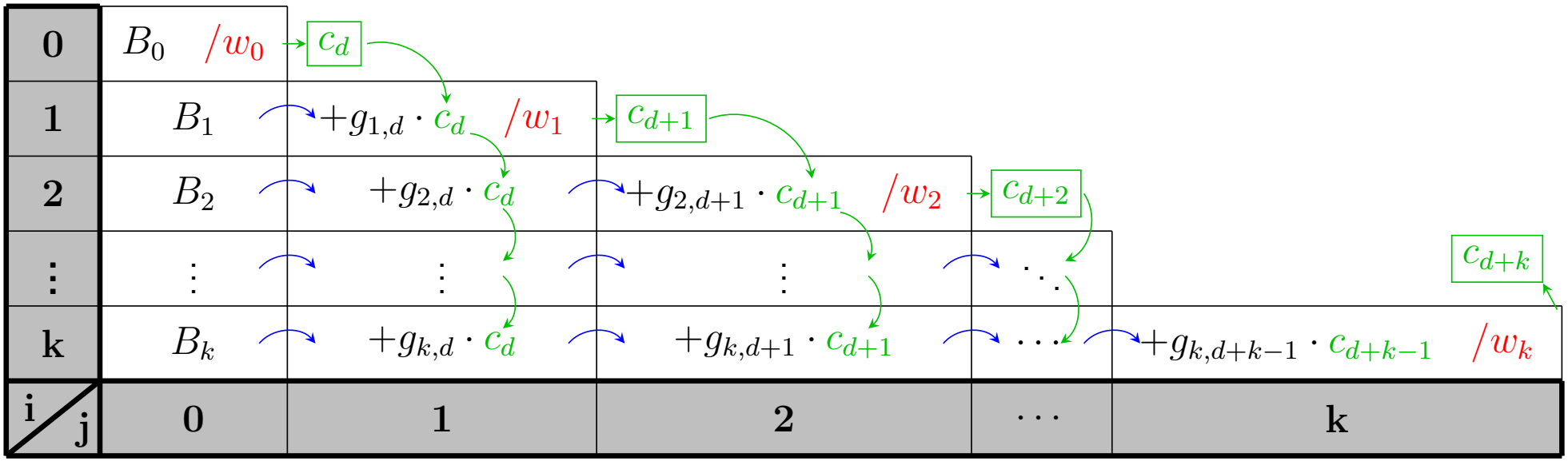


Figure 3: The T Table

For $j \leq i$:

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

If $i = j$:

$$T(i,j) := T(i,j)/w_i$$
$$c_{d+i} := T(i,j)$$

Tiling Strategy for the T Table

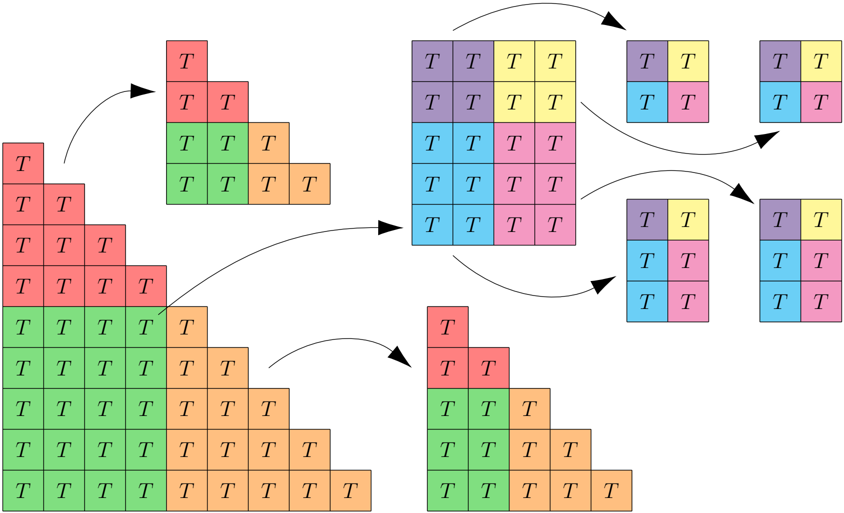


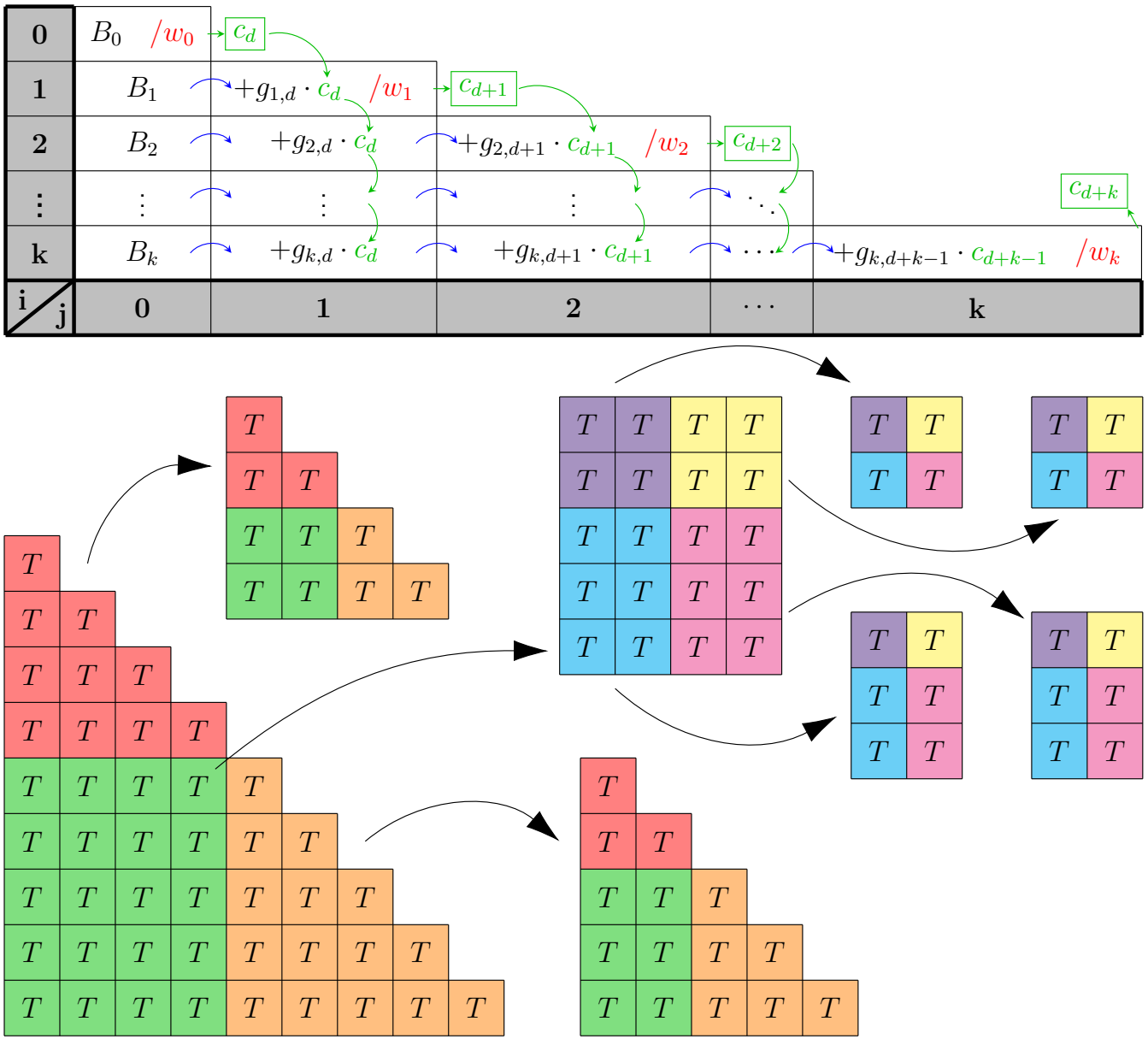
Figure 4: Recursively Dividing the T Table

What to Parallelize

When calculating $k+1$ coefficients $(c_d, \ldots, c_{d+k})$:

1. Calculate g table (**Map** pattern)
2. Calculate T table

Parallelizing the T Table Computation



Experimental Setup

Implemented (ODE orders 2, 8, 32):

- In MAPLE (*dsolve, 'series'*)
- In BPAS: Iterative, Tiling, Parallel

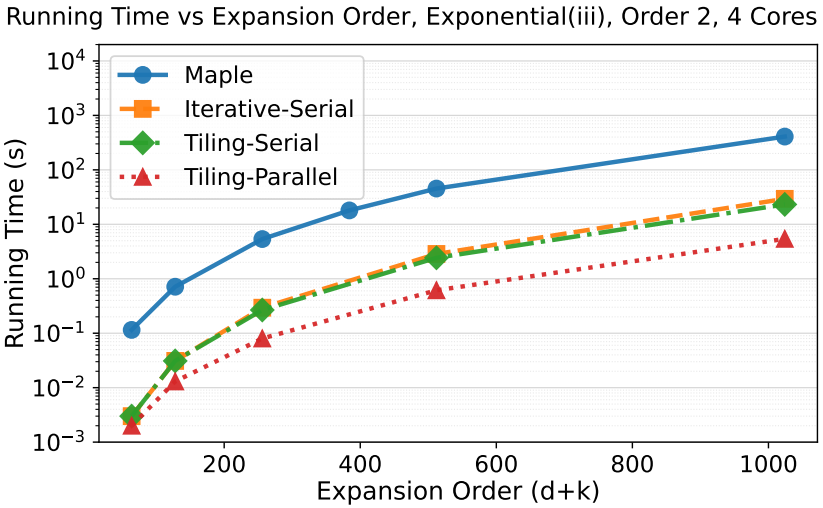
Machines (support SMT):

- 4 Core Intel i7, 2×8GB RAM
  - MAPLE, Iterative, Tiling, Parallel
- 64 Core AMD EPYC, 24×32GB RAM
  - Iterative, Tiling, Parallel

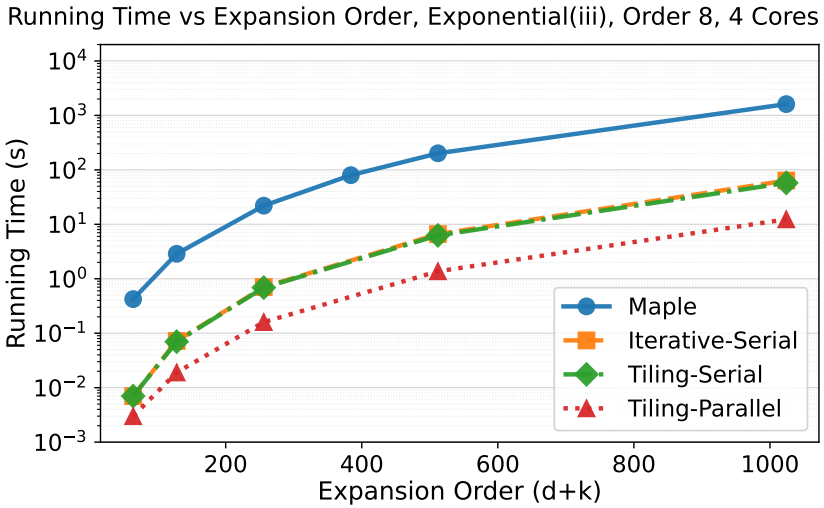
[Exponential Coefficients] Family (*iii*):

- $\sum_{i=0}^{n}(i+1)e^{(i+1)x}y^{(i)}(x) = e^{nx}$
- $3e^{3x}y''(x) + 2e^{2x}y'(x) + e^{x}y(x) = e^{2x}$

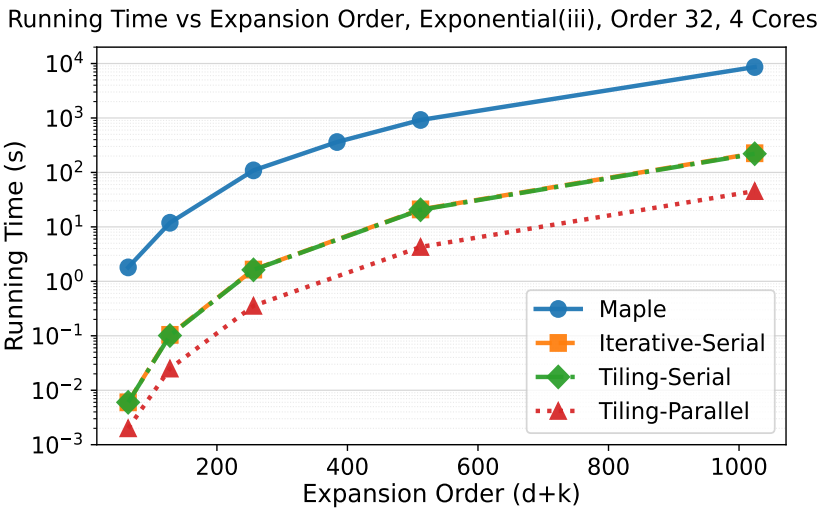
Selected Results - Exponential Family, 4 Cores



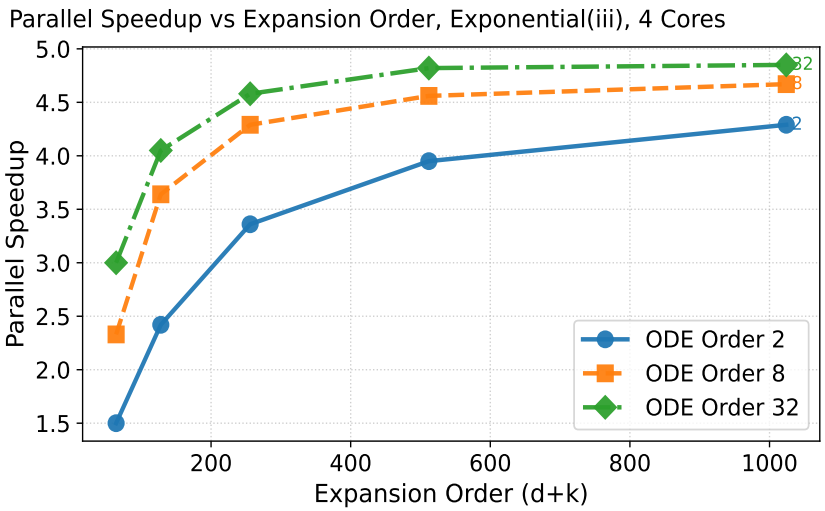
(a) Exponential ODE, Order 2.



(b) Exponential ODE, Order 8.



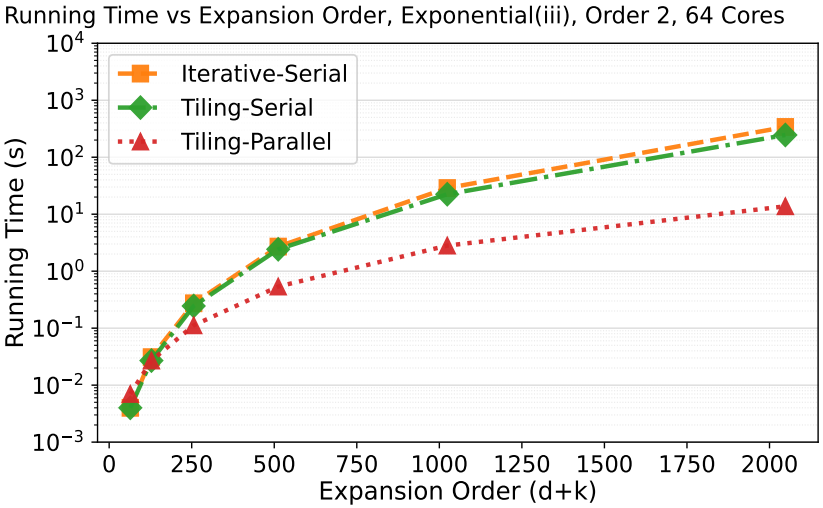
(c) Exponential ODE, Order 32.



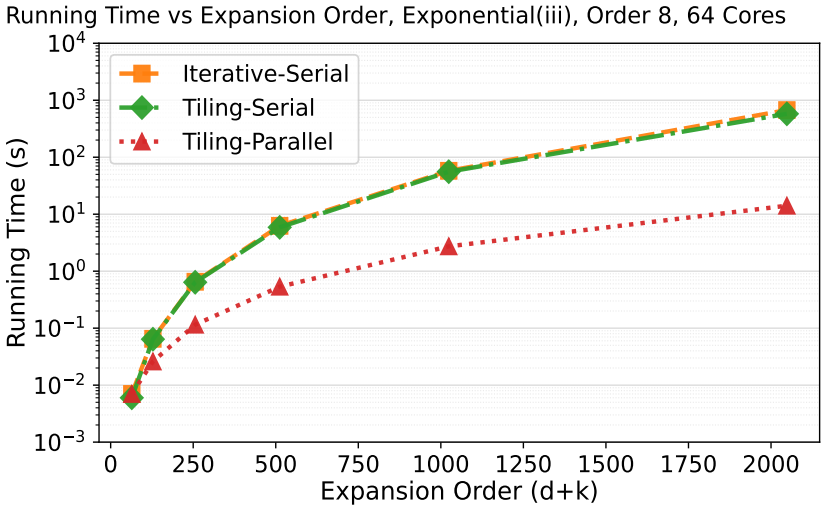
(d) Parallel Speedup, Exponential ODE.

Figure 5: 4 Cores - Experimental Results - Exponential ODE Family (iii).

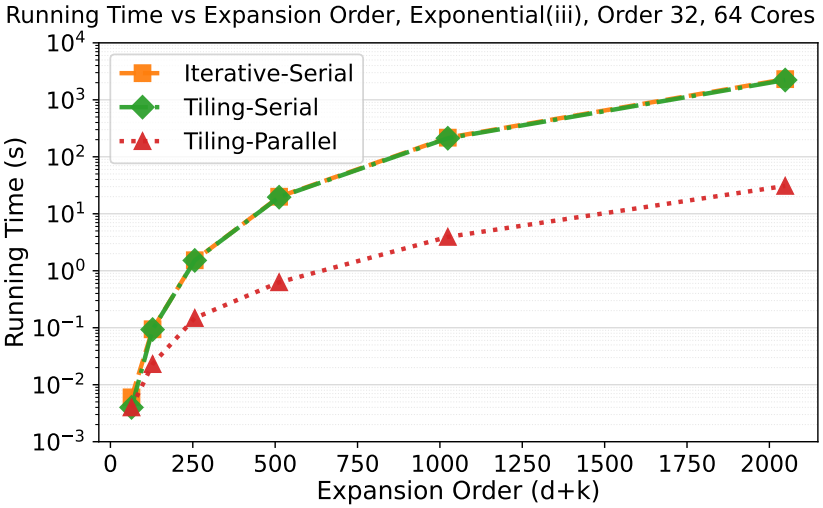
Selected Results - Exponential Family, 64 Cores



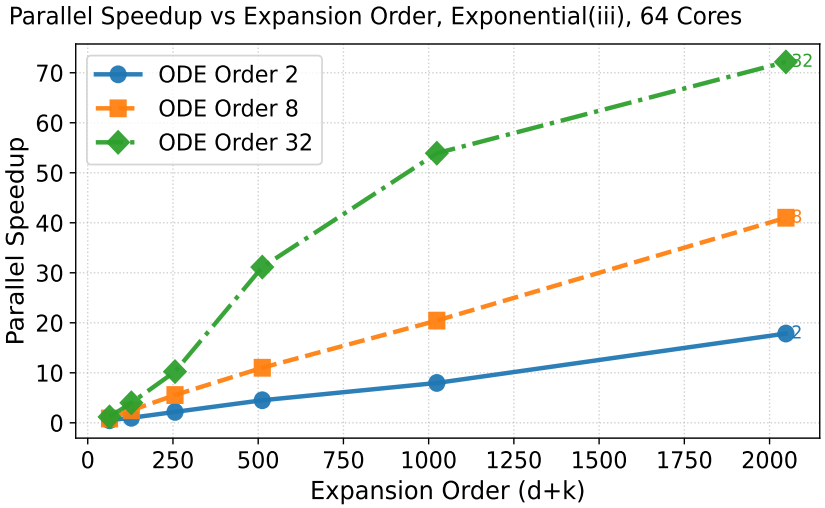
(a) Exponential ODE, Order 2.



(b) Exponential ODE, Order 8.



(c) Exponential ODE, Order 32.



(d) Parallel Speedup, Exponential ODE.

Figure 6: 64 Cores - Experimental Results - Exponential ODE Family (iii).

End

Thank you for listening!

Questions?

[Alvandi et al., 2020, Brandt and Moreno Maza, 2021, Solis-Reyes et al., 2025]

References I

[Alvandi et al., 2020] Alvandi, P., Ataei, M., Kazemi, M., and Moreno Maza, M. (2020).
On the extended Hensel construction and its application to the computation of real limit points.
J. Symb. Comput., 98:120–162.

[Brandt and Moreno Maza, 2021] Brandt, A. and Moreno Maza, M. (2021).
On the complexity and parallel implementation of Hensel's lemma and Weierstrass preparation.
In *Computer Algebra in Scientific Computing (CASC '21)*, volume 12865 of *LNCS*, pages 78–99. Springer.

References II

[Granlund et al., 2025] Granlund, T., Sjödin, G., and the GMP development team (2025).
GNU MP: The GNU Multiple Precision Arithmetic Library (version 6.2.0).
http://gmplib.org.

[Kung and Traub, 1978] Kung, H. T. and Traub, J. F. (1978).
All algebraic functions can be computed fast.
J. ACM, 25(2):245–260.

References III

[McCool et al., 2012a] McCool, M., Reinders, J., and Robison, A. (2012a).
Structured parallel programming: patterns for efficient computation.
Elsevier.

[McCool et al., 2012b] McCool, M. D., Robison, A. D., and Reinders, J. (2012b).
Structured parallel programing : patterns for efficient computation.

Elsevier, Morgan Kaufmann, Amsterdam.

References IV

[Sasaki and Kako, 1999] Sasaki, T. and Kako, F. (1999).
Solving multivariate algebraic equation by Hensel construction.
Japan J. Indust. and Appl. Math.

[Solis-Reyes et al., 2025] Solis-Reyes, G. A., Moreno Maza, M., and Brandt, A. (2025).
Parallel computation of the power series solutions to linear ordinary differential equations.
In *Computer Algebra in Scientific Computing*, pages 380–400. Springer Nature Switzerland.

[van der Hoeven, 2002] van der Hoeven, J. (2002).
Relax, but don't be too lazy.
J. Symb. Comput., 34(6):479–542.

References V

[Wikipedia, 2025] Wikipedia (2025).
Thread pool.
https://en.wikipedia.org/wiki/Thread_pool.