Efficient Evaluation of Large Polynomials

Liyun Li, Charles E. Leiserson, Marc Moreno Maza, Yuzhen Xie

September 17, 2010

Introductory example (1/2)

Consider Newton's root finding method for approximating a solution of $f(x) = 0$ starting from an initial guess $x = x_0$:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

This iteration continues until $f(x_n)$ satisfies a specified tolerance.

$$\begin{array}{llll} x_0 & f(x_0) & f'(x_0) & \Rightarrow \quad x_1 \\ x_1 & f(x_1) & f'(x_1) & \Rightarrow \quad x_2 \\ & & & \vdots \\ x_{n-1} & f(x_{n-1}) & f'(x_{n-1}) & \Rightarrow \quad x_n \end{array}$$

Introductory example (2/2)

Suppose that

- f is encoded by a SLP P (compiled code) of length ℓ.
- to be evaluated at m points in dimension n with `int` coordinates
- using a (Z, L) data cahce and (Z', L') instruction cache.

If ℓ and m *are large enough* ($Z' \ll \ell$ and $\ell L \ll nmZ'$) and n *is small enough* ($n \ll LL'$) one can prove that this process will incur

- $O(m\,\ell/L' + m)$ instruction cache misses and
- $O(mn/L + 1)$ data cache misses.

If P is decomposed $s \simeq \ell/Z'$ subprograms then the total number of cache misses drops to $O((\ell/Z')(mn/L) + \ell/L')$.

This is indeed better since we have:

$$(\ell/Z')(mn/L) \ll m\,\ell/L'.$$

Objective 1

Given a polynomial f expressed by the sum of its terms, find a representation of f that "minmizes" the evaluation cost of f.

Observe that:

- The evaluation points and their number are not known in advance.
- The evaluation points may be symbolic expressions.

Input (557 operations):

$$a^2b^2cde^2ghjlm + a^2b^2cdeh^2klmn + a^2bcdegh^2lm^2n + ab^3c^2de^2ghij +$$
$$ab^3c^2deh^2ikn + ab^2c^2de^2ghjlm + ab^2c^2degh^2imn + ab^2c^2deh^2klmn +$$
$$ab^2cd^2e^3ghjm + ab^2cd^2e^2h^2kmn + abc^2degh^2lm^2n + abcd^2e^2gh^2m^2n +$$
$$b^3c^3de^2ghij + b^3c^3deh^2ikn + b^2c^3degh^2imn + b^2c^2d^2e^3ghjm +$$
$$b^2c^2d^2e^2h^2kmn + bc^2d^2e^2gh^2m^2n + ab^2ceghij^2lm + ab^2ch^2ijklmn +$$
$$abc^2de^2ghjlm + abc^2deh^2klmn + abcgh^2ijlm^2n + abe^2ghjklm^2n +$$
$$abeh^2k^2lm^2n^2 + ac^2degh^2lm^2n + aegh^2klm^3n^2 + b^3c^2eghi^2j^2 +$$
$$b^3c^2h^2i^2jkn + b^2c^3de^2ghij + b^2c^3deh^2ikn + b^2c^2gh^2i^2jmn +$$
$$b^2cde^2ghij^2m + b^2cdeh^2ijkmn + b^2ce^2ghijkmn + b^2ceh^2ik^2mn^2 +$$
$$bc^3degh^2imn + bc^2d^2e^3ghjm + bc^2d^2e^2h^2kmn + bcdegh^2ijm^2n +$$
$$bcegh^2ikm^2n^2 + bde^3ghjkm^2n + bde^2h^2k^2m^2n^2 + c^2d^2e^2gh^2m^2n +$$
$$de^2gh^2km^3n^2$$

Applying Multivariate Horner scheme of Maple (382 operations)

$$de^2gh^2km^3n^2 + c^2d^2e^2gh^2m^2n + ((k^2m^2n^2h^2 + jkm^2nghe)e^2d +$$
$$(egh^2ikm^2n^2 + degh^2ijm^2n + ((h^2m^2ng + mh^2kn + jmghe)e^2d^2 +$$
$$dih^2nmegc)c)c + (((k^2n^2imh^2 + jiknmghe)e + (ijmh^2kn + ij^2mghe)ed +$$
$$(i^2jgh^2mn + (mh^2kn + jmghe)e^2d^2 + (ih^2nmg + ih^2kn + ijghe)edc)c)c +$$
$$(i^2j^2egh + i^2jh^2kn + (ih^2kn + ijghe)edc)c^2b)b)b + (c^2degh^2lm^2n + egh^2klm^3n^2 +$$
$$((lk^2m^2n^2h^2 + jkm^2nlghe)e + (h^2m^2nd^2e^2g + lm^2ijgh^2n + (h^2m^2nlg + lmh^2kn +$$
$$jlmghe)edc)c + ((ij^2lmegh + jlimh^2kn + (mh^2kn + jmghe)e^2d^2 + (ih^2nmg +$$
$$lmh^2kn + jlmghe)edc)c + (ih^2kn + ijghe)edc^2b)b)b + (h^2m^2ndcleg + (lmh^2kn +$$
$$jlmghe)edcb)ba)a$$

Applying Maple's optimize command with 'tryhard' option (153 operations)

$$t_{86} = al, t_{85} = de, t_{84} = ij, t_{83} = kn, t_{55} = c^2, t_{82} = at_{55}, t_{46} = m^2, t_{81} = bt_{46},$$
$$t_{57} = b^2, t_{80} = ct_{57}, t_{44} = n^2, t_{79} = kt_{44}, t_{78} = t_{57}m, t_{77} = a^2l, t_{76} = t_{57} + b,$$
$$t_{52} = e^2, t_{53} = d^2, t_{75} = t_{53}t_{52}, t_{56} = bt_{57}, t_{74} = t_{55}t_{56}, t_{73} = et_{86}, t_{72} = it_{78},$$
$$t_{71} = t_{55}t_{75}, t_{49} = i^2, t_{70} = t_{49}t_{74}, t_{69} = t_{81}t_{83}, t_{68} = ct_{72}, t_{67} = it_{52}t_{80}, t_{54} = ct_{55},$$
$$t_{66} = (t_{56} + t_{57})t_{54}, t_{65} = t_{84} + t_{77}, t_{64} = (t_{75} + lt_{84})a, t_{63} = (dt_{73} + t_{75})t_{55},$$
$$t_{51} = et_{52}, t_{50} = h^2, t_{48} = j^2, t_{45} = mt_{46},$$
$$t1 = ((et_{68} + (t_{73} + dt_{52})t_{81})k^2t_{44} + (jt_{70} + (at_{74} + t_{66})it_{85} + (bt_{63} + (t_{63} +$$
$$(t_{65}t_{85} + t_{64})c)t_{57})m)t_{83})t_{50} + (((t_{45}t_{86} + cit_{81})et_{79} + (t_{55}t_{49}jt_{78} + (t_{71} +$$
$$(t_{71} + ct_{64})b)t_{46})n)t_{50} + ((t_{68}t_{86} + t_{70})t_{48}e + (t_{52}t_{69}t_{86} + (t_{67}t_{83} + (at_{80} +$$
$$t_{76}t_{55})t_{53}t_{51})m)j)h + ((t_{52}t_{45}t_{79} + (t_{54}t_{72} + (lt_{46} + t_{72})t_{82} + (t_{54}im + (lt_{82} +$$
$$t_{65}c)t_{46})b)ne)t_{50} + (t_{48}mt_{67} + (t_{51}t_{69} + (ct_{77}t_{78} + it_{66} + (t_{56}i + t_{76}ml)t_{82})t_{52})j)h)d)g$$

Our output (142 operations)

$$(((((bej + hmn)d(ab + bc + c) + b^2ij^2)c + (bej + hmn)kmn)(al + de)+$$
$$(bej + hmn)bcikn + (c(ab + bc + c) + jm)bcdihn)m + (bij + de(ab+$$
$$bc + c))b^2c^2ij)egh + ((((((al + de)(ab + bc + c) + bij)d + knbi)e + ajlbi)c+$$
$$(al + de)kmne)m + (bij + de(ab + bc + c))bc^2i)bh^2kn + (alm + bci)bcijgh^2mn$$

After application of common subexpression elimination technique to our output (94 operations)

$$t_5 = bej + hmn, t_8 = bc, t_9 = ab + t_8 + c, t_{11} = b^2, t_{13} = j^2, t_{18} = mn,$$
$$t_{21} = al, t_{22} = de, t_{23} = t_{21} + t_{22}, t_{41} = bi, t_{42} = t_{41}j, t_{44} = t_{42} + t_{22}t_9,$$
$$t_{46} = c^2, t_{47} = t_{46}i, t_{76} = h^2,$$
$$t_{91} = ((((t_5dt_9 + t_{11}it_{13})c + t_5kt_{18})t_{23} + t_5bcikn + (ct_9 + jm)bcdihn)m +$$
$$t_{44}t_{11}t_{47}j)egh + ((((t_{23}t_9 + t_{42})d + knt_{41})e + ajlbi)c + t_{23}kt_{18}e)m +$$
$$t_{44}bt_{47})bt_{76}kn + (t_{21}m + t_8i)bcijgt_{76}mn$$

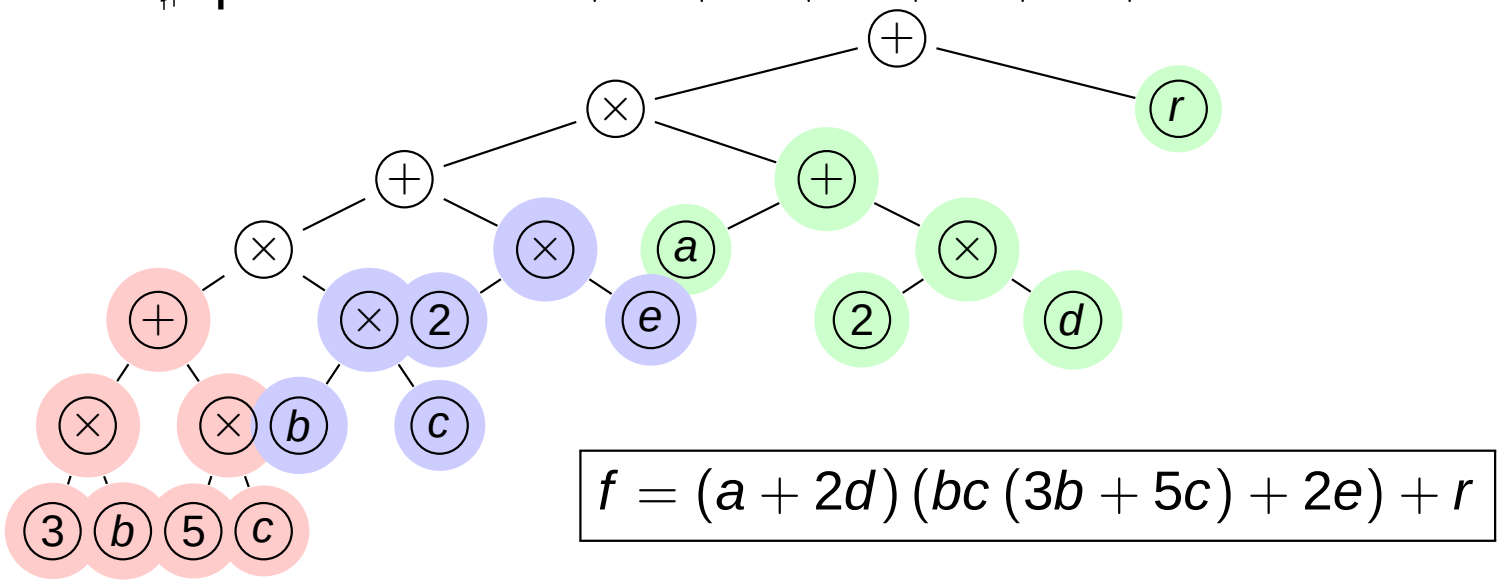
Objective 2

From our "slim representation" of f deduce a schedule for evaluating f efficiently in terms of data locality and parallelism.

Example

$$f = 3ab^2c + 5abc^2 + 2ae + 6b^2cd + 10bc^2d + 4de + r$$

$$\sharp\text{operations} = 4 + 4 + 2 + 4 + 4 + 2 + 6 = 30$$



$$f = (a + 2d)(bc(3b + 5c) + 2e) + r$$

Figure: An evaluation schedule of f when 3 processors are available

Main results

- Given a polynomial in expanded form, compute a syntactic decomposition of it which permits a cheaper evaluation.

Go to section details

- Generate Cilk++ program for parallel evaluation of a syntactic decomposition.

Go to section details

- Parallelize the computation of syntactic decomposition and of the minimal elements of a partially ordered set.

Go to section details

Return to Outline

“Minimizing” the evaluation cost

- Background
- The hypergraph method
- Complexity estimation

Evaluating univariate polynomials

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- The most direct evaluation computes terms one by one.

$$\begin{array}{rcccccccccc} \sharp\text{mult} & = & 0 & + & 1 & + & 2 & + \cdots + & n & = & n(n+1)/2 \\ \sharp\text{add} & = & 0 & + & 1 & + & 1 & + \cdots + & 1 & = & n \end{array}$$

- Horner's rule writes the polynomial in the form

$$\begin{array}{rcccccccccc} f(x) & = & a_0 & + & x(a_1 & + & x(a_2 & + \cdots + & x(a_{n-1} & + & a_n x))) \\ \sharp\text{mult} & = & 0 & + & 1 & + & 1 & + \cdots + & 1 & + & 1 = n \\ \sharp\text{add} & = & 0 & + & 1 & + & 1 & + \cdots + & 1 & + & 1 = n \end{array}$$

- Horner's rule is optimal for evaluating arbitrary univariate polynomials.

Multivariate Horner's rule and friends

There are no known methods proven to be optimal! However there is an abundant literature on related topics.

- *Generation of Optimal Code for Expressions via Factorization* (Melvin A. Breuer, 1969)
- *On the Multivariate Horner Scheme* (J. M. Peña, 2000)
- *Greedy Algorithms for Optimizing Multivariate Horner Schemes* (M. Ceberio & V. Kreinovich, 2004)
- *Optimizing Polynomial Expressions by Algebraic Factorization and Common Subexpression Elimination* (A. Hosangadi, F. Fallah & R. Kastner, 2006)

Common subexpression elimination

Common subexpression elimination (CSE) is a compiler optimization technique that searches for instances of identical expressions.

Example

Consider this polynomial,

$$f = (ab + c)d + (ab + d)e$$

It can be rewritten in a CSE form as

$$\begin{aligned} g &= ab \\ f &= (g + c)d + (g + d)e \end{aligned}$$

It can always be used as a post-processing technique.

Functional decomposition (and algebraic factorization)

$$f = g \circ h$$

Example

$$\begin{aligned} f &= x^9 + 3x^7 + 4x^6 + 3x^5 + 8x^4 + 6x^3 + 4x^2 + 5x + 3 \\ g &= x^3 + x^2 + 1 \\ h &= x^3 + x + 1 \end{aligned}$$

Weakness

It was shown[a] that "most" polynomials over an arbitrary field are indecomposable.

[a] *Some Results on the Functional Decomposition of Polynomials*, Mark William Giesbrecht, 1988.

Notations

- terms(f) refers to the set of all the terms of f.
- mons(f) refers to the set of all the monomials of f.
- Writing $f \in \mathbb{K}[X]$ indicates that the coefficients of the polynomial f are in base field $\mathbb{K}$ and that its variables are in set X.
- syntactic product : $g\,h$ is a syntactic product if there is no grouping or cancellation of terms when multiplying g and h, written as $g \odot h$.

$(a+b)(a-b)$	cancellation	$ab - ab$
$(a+b)(a+b)$	grouping	$ab + ab$
$(a+b)(c+d)$	syntactic product	$(a+b) \odot (c+d)$

- syntactic sum : $g + h$ is a syntactic sum if there is no grouping or cancellation of terms when adding g and h, written as $g \oplus h$.

Syntactic Factorization

For $f, g, h \in \mathbb{K}[X]$, we say that $g\,h$ is a *syntactic factorization* of f if

$$f = g \odot h.$$

Example

$(x-1)(x^2+x+1)$ is not a syntactic factorization of x^3-1.

Property

$$f = g \odot h \Rightarrow \begin{cases} \text{degree}(f) & = & \text{degree}(g) & + & \text{degree}(h); \\ \sharp\text{terms}(f) & = & \sharp\text{terms}(g) & \times & \sharp\text{terms}(h). \end{cases}$$

Partial Syntactic Factorization

A set of pairs of polynomials $(g_1, h_1), (g_2, h_2), \ldots, (g_e, h_e)$ and a polynomial r in $\mathbb{K}[X]$ is a *partial syntactic factorization* of f if:

1. $f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus \cdots \oplus (g_e \odot h_e) \oplus r$
2. r or any part of r does not admit any nontrivial syntactic factorization.

Example

A partial syntactic factorization of

$$f = ac + ad + bc + bd + e$$

is

$$(a + b) \odot (c + d) \oplus e$$

Partial Syntactic Factorization

Example

$$f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus (g_3 \odot h_3) \oplus (g_4 \odot h_4) \oplus r$$

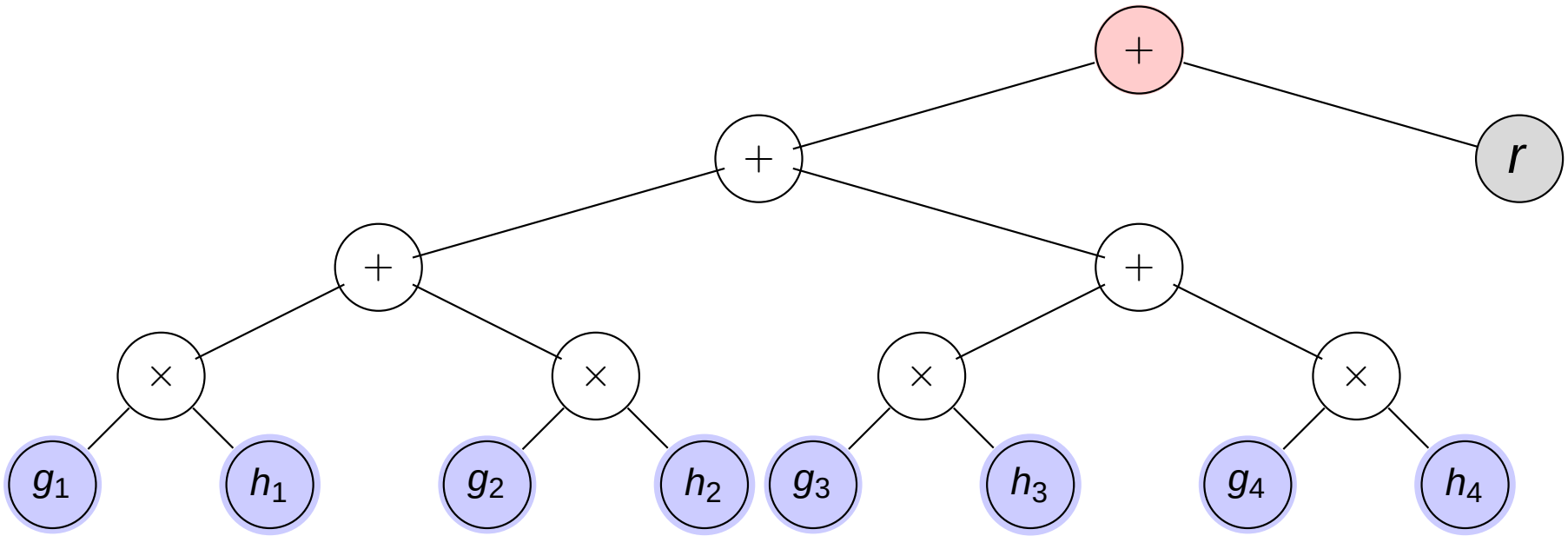


Figure: syntactic decomposition

Base monomial set $\mathcal{M}$: given by an oracle

Question

How do we discover part of f that admits a syntactic factorization?

Example

$$f = ac + ad + bc + bd + e = (a + b)(c + d) + e$$

If the monomial set $\{a, b\}$ is given, we can find $(a + b) \odot (c + d)$.

We say a partial syntactic factorization

$$f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus \cdots \oplus (g_e \odot h_e) \oplus r$$

is w.r.t a base monomial set $\mathcal{M}$ if $\text{mons}(g_i) \in \mathcal{M}$ for all $i = 1, \ldots, e$.

Question

How to exploit the base monomial set to build a syntactic factorization?

Using a hypergraph designed for it!

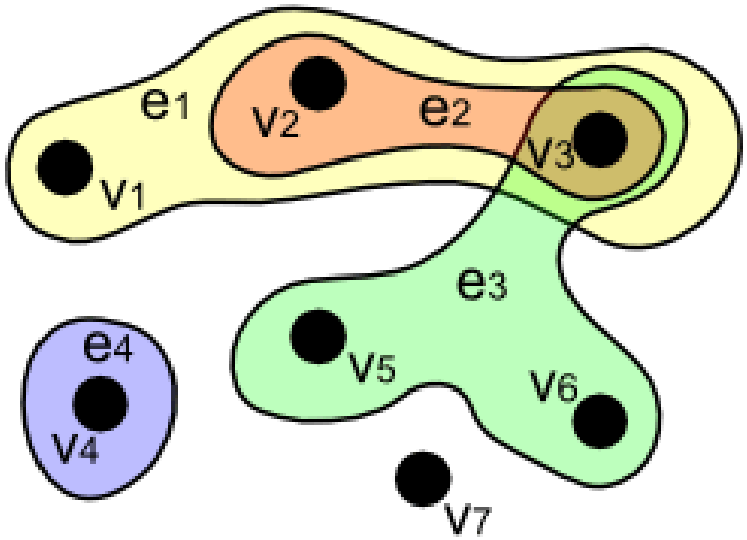
Hypergraph review

A hypergraph is a generalization of a graph where an edge can connect any number of vertices. Formally, a hypergraph H is a pair $H = (\mathcal{V}, \mathcal{E})$ where

- $\mathcal{V}$ is a set of elements, called vertices;
- $\mathcal{E}$ is a set of non-empty subsets of $\mathcal{V}$, called hyperedges.

$$\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$$

$$\mathcal{E} = \begin{cases} e_1 = \{v_1, v_2, v_3\} \\ e_2 = \{v_2, v_3\} \\ e_3 = \{v_3, v_5, v_6\} \\ e_4 = \{v_4\} \end{cases}$$



Hypergraph HG($f, \mathcal{M}$)

Given f and $\mathcal{M}$, then HG$(f, \mathcal{M}) = (\mathcal{V}, \mathcal{E})$ is defined as

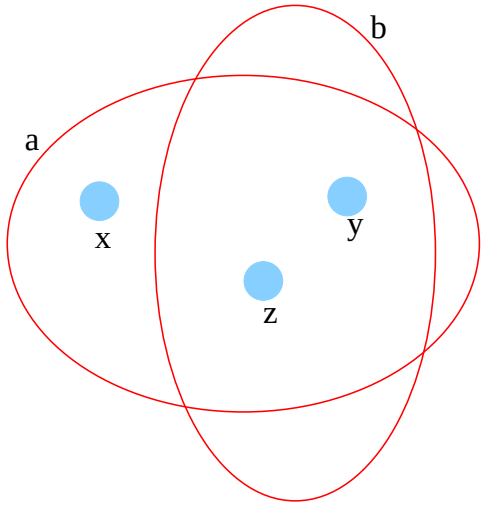
- $\mathcal{V} = \mathcal{M}$
- $\mathcal{E} = \{E_q \mid E_q \text{ is nonempty}\}$, where E_q denotes the set

$$\{m \in \mathcal{M} \mid m\,q \in \text{mons}(f)\}.$$

for an arbitrary monomial q.

Example

$$\begin{aligned} f &= ax + ay + az + by + bz \\ \mathcal{M} &= \{x, \ y, \ z\} \\ \mathcal{V} &= \{x, \ y, \ z\} \\ \mathcal{E} &= \begin{cases} E_a = \{x, \ y, \ z\}, \\ E_b = \{y, \ z\}. \end{cases} \end{aligned}$$



Property of hypergraph HG($f, \mathcal{M}$)

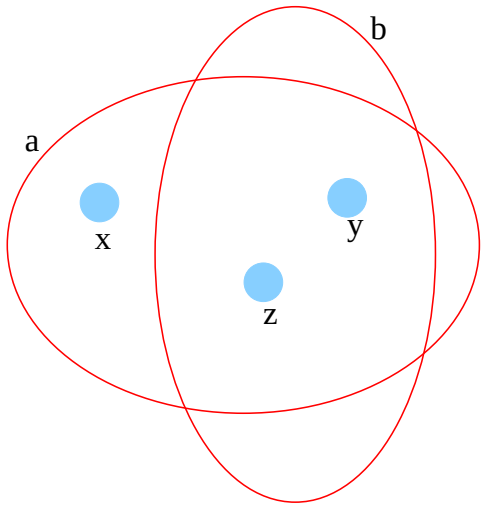
$$\begin{cases} f = g \odot h + \cdots \\ \text{mons}(g) \subseteq \mathcal{M} \end{cases} \Rightarrow \text{mons}(g) \subseteq \cap_{q \in \text{mons}(h)} E_q.$$

Example

$$\begin{aligned} f &= ax + ay + az + by + bz \\ &= (y + z) \odot (a + b) + ax \end{aligned}$$

$$\mathcal{V} = \{x, \quad y, \quad z\}$$

$$\mathcal{E} = \begin{cases} E_a = \{x, \quad y, \quad z\}, \\ E_b = \{y, \quad z\}. \end{cases}$$



Constructing Syntactic Factorization

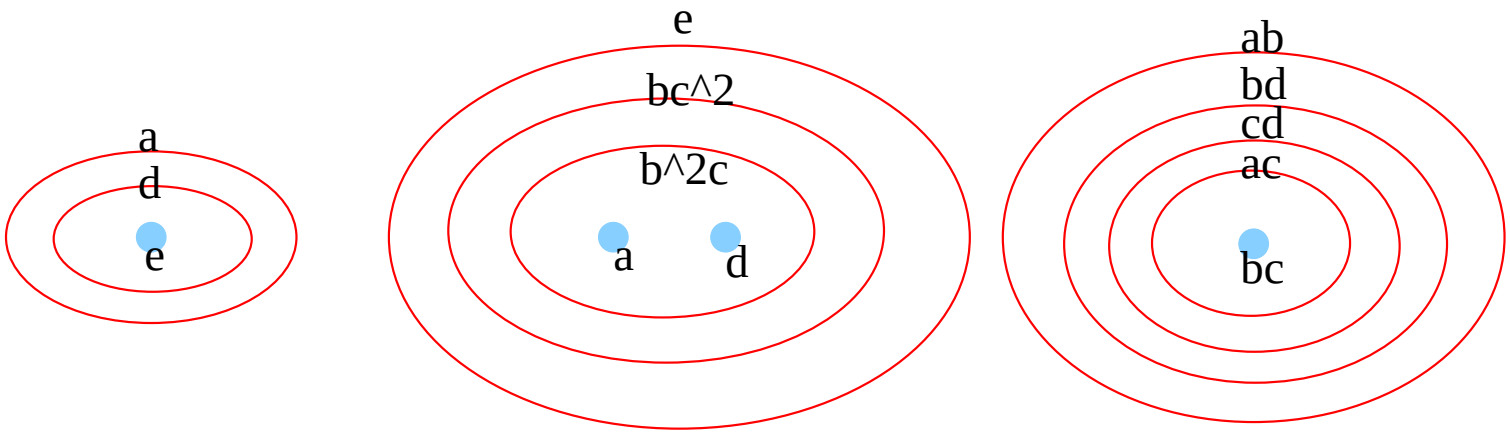
Question

How to construct a syntactic factorization from a hypergraph?

Greedy strategy

- find the largest intersection of edges in the hypergraph.
- construct a candidate syntactic factorization from it.
- build a system to solve the coefficients in a candidate syntactic factorization.

Constructing Syntactic Factorization: example



- $f = 3ab^2c + 5abc^2 + 2ae + 6b^2cd + 10bc^2d + 4de + r$
- $\mathcal{M} = \{a, bc, d, e\}$
- $E_{b^2c} \cap E_{bc^2} \cap E_e = \{a, d\}$
- the candidate syntactic factorization is therefore

$$\begin{aligned} & (\alpha_1 a + \alpha_2 d)(\beta_1 b^2c + \beta_2 bc^2 + \beta_3 e) \\ = \quad & 3ab^2c + 5abc^2 + 2ae + 6b^2cd + 10bc^2d + 4de \end{aligned}$$

Constructing Syntactic Factorization: example

$$
\begin{aligned}
& (\alpha_1 a + \alpha_2 d)(\beta_1 b^2 c + \beta_2 b c^2 + \beta_3 e) \\
= \quad & 3ab^2c + 5abc^2 + 2ae + 6b^2cd + 10bc^2d + 4de
\end{aligned}
$$

$$
\begin{cases}
\alpha_1\beta_1 = 3 \\
\alpha_1\beta_2 = 5 \\
\alpha_1\beta_3 = 2 \\
\alpha_2\beta_1 = 6
\end{cases}
\overset{\alpha_1=1}{\Longrightarrow}
\begin{cases}
\alpha_1 = 1 \\
\alpha_2 = 2 \\
\beta_1 = 3 \\
\beta_2 = 5 \\
\beta_3 = 2
\end{cases}
\Rightarrow
\begin{cases}
\alpha_2\beta_2 = 10 \\
\alpha_2\beta_3 = 4
\end{cases}
$$

A syntactic factorization has been found:

$$(a + 2d) \odot (3b^2c + 5bc^2 + 2e)$$

Constructing partial syntactic factorization

We have an algorithm to construct a syntactic factorization of part of f.

Question

How to build a partial syntactic factorization of f?

$$f = (g_1 \odot h_1) \oplus (f - g_1 h_1)$$

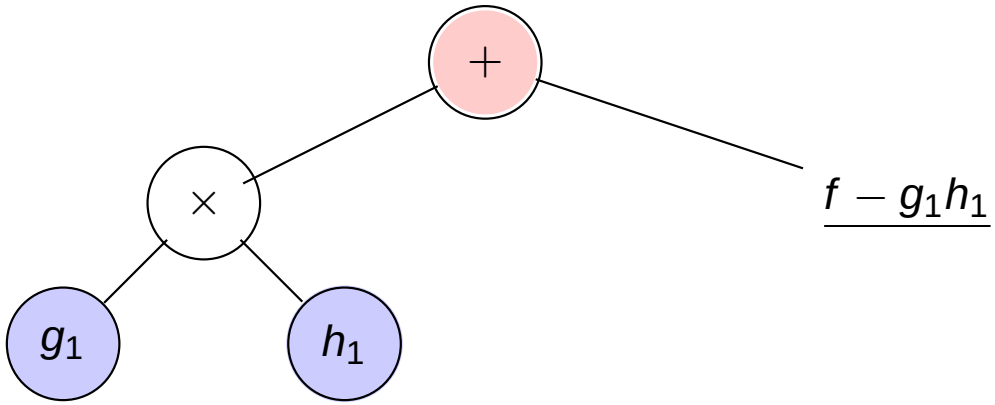


Figure: computation of partial syntactic factorization

Constructing Partial Syntactic Factorization

$$f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus (f - g_1 h_1 - g_2 h_2)$$

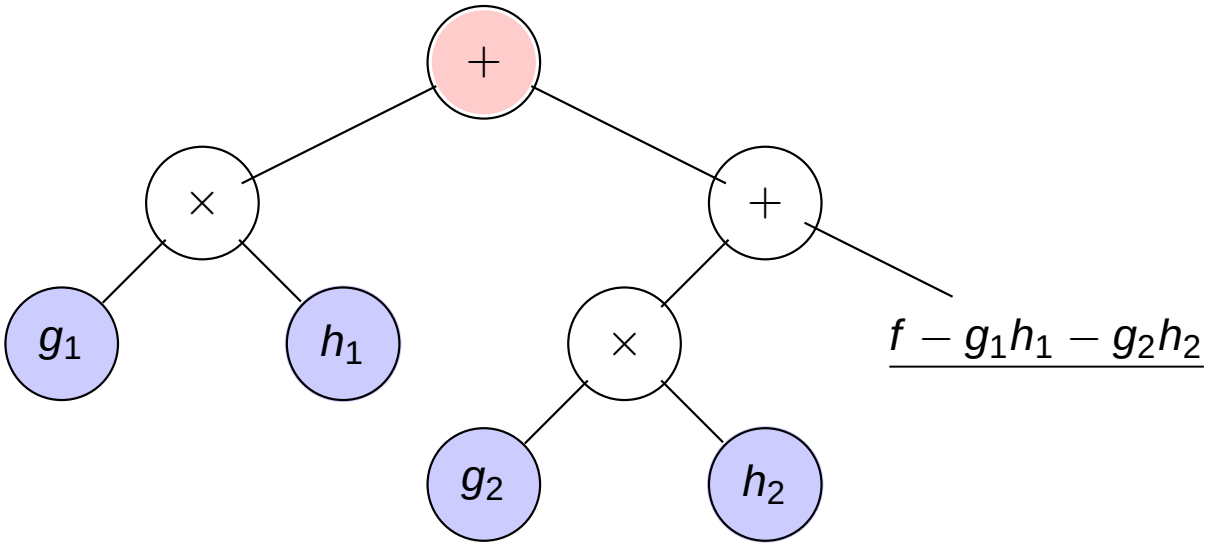


Figure: computation of partial syntactic factorization

Constructing Partial Syntactic Factorization

$$f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus (g_3 \odot h_3) \oplus (f - g_1h_1 - g_2h_2 - g_3h_3)$$

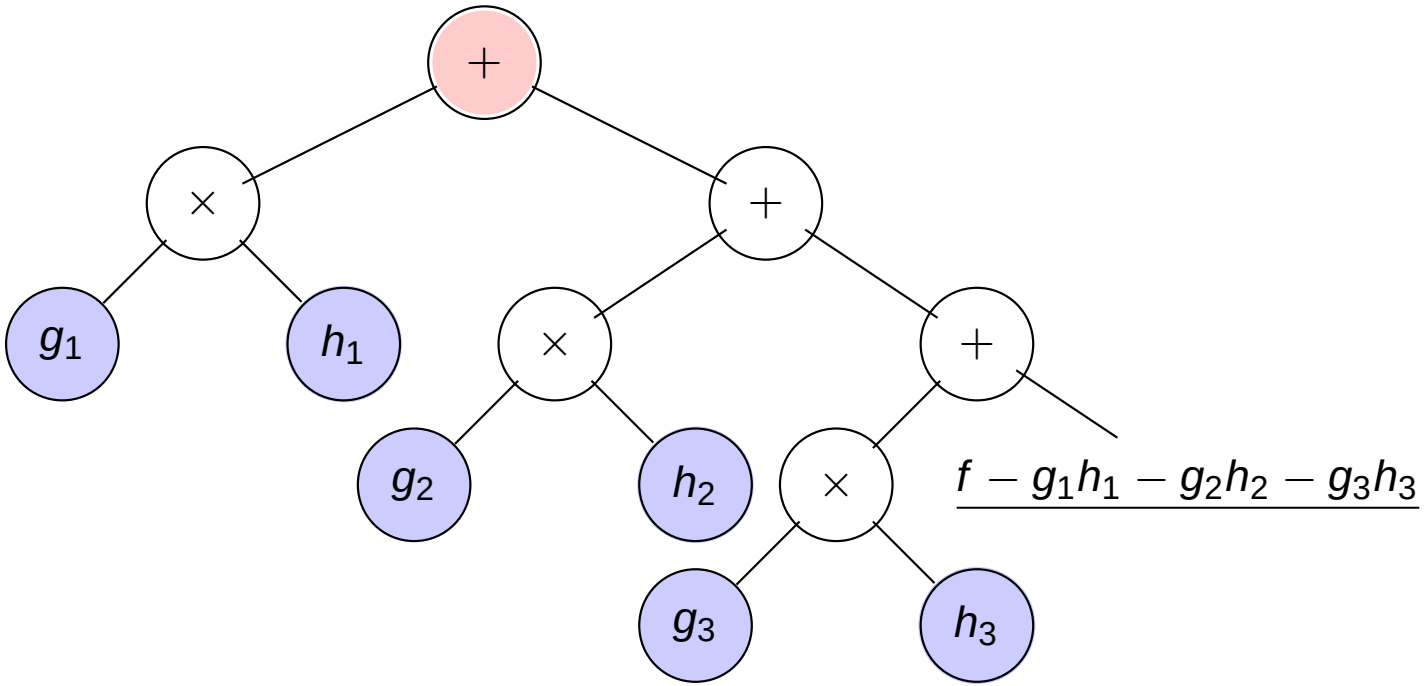


Figure: computation of partial syntactic factorization

Constructing Partial Syntactic Factorization

$$f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus (g_3 \odot h_3) \oplus r$$

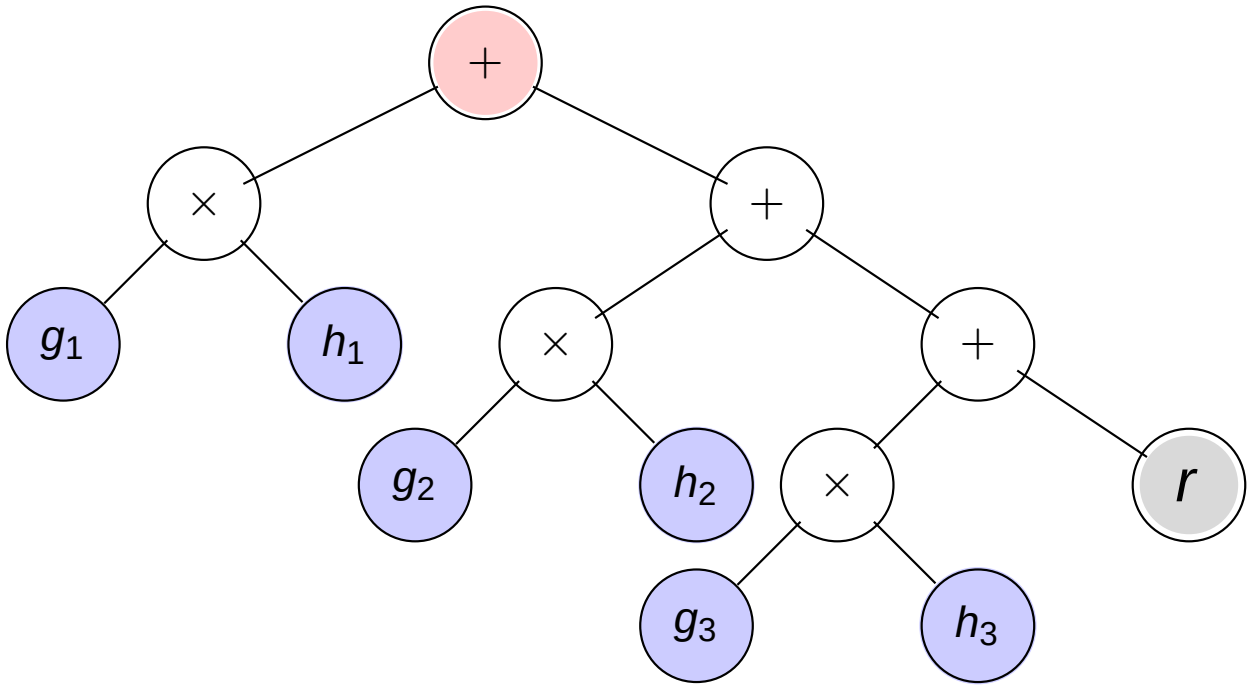
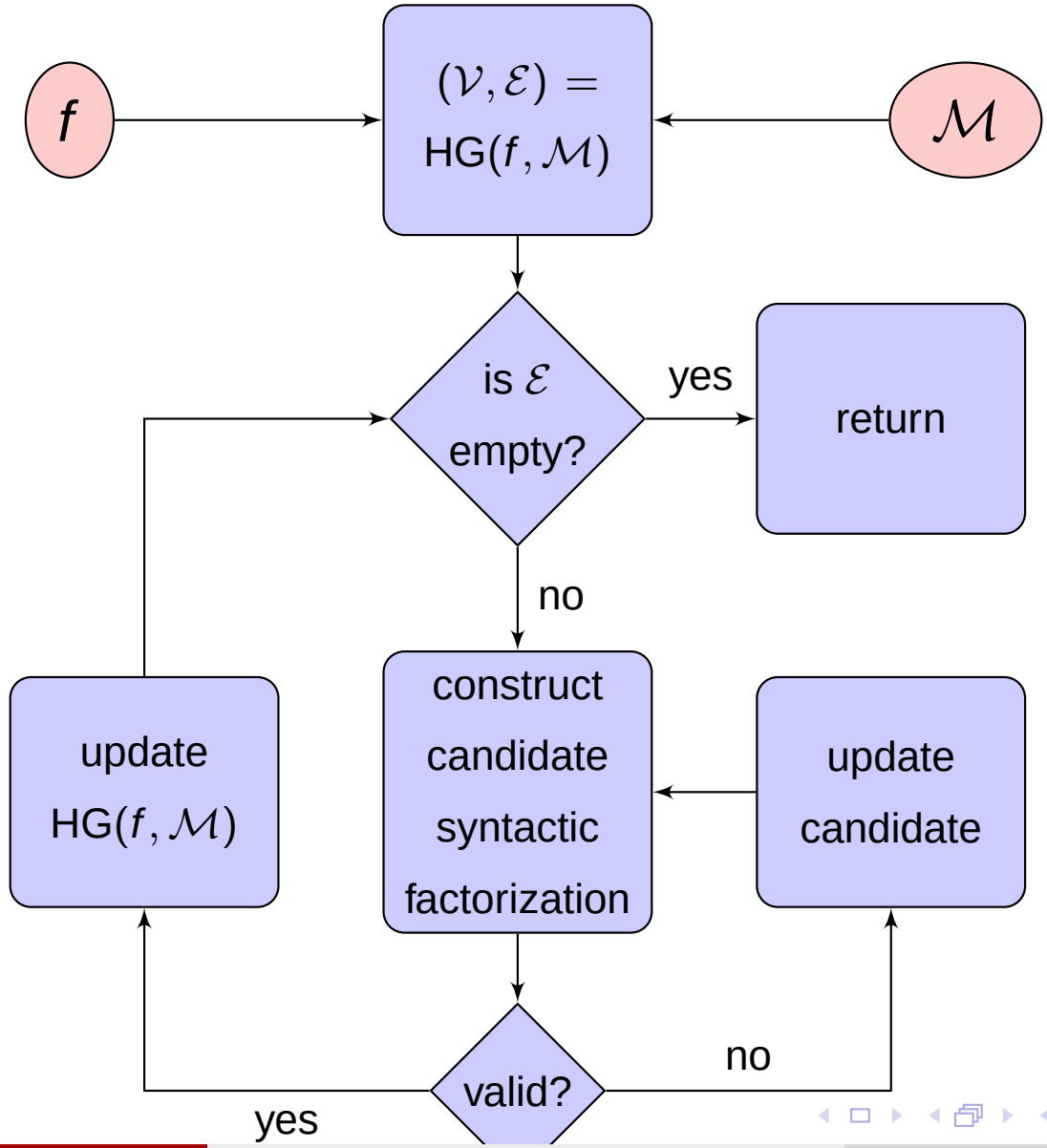


Figure: computation of partial syntactic factorization

Flow chart: partial syntactic factorization



About base monomial set (1/2)

Question

How to find an appropriate base monomial set $\mathcal{M}$?

Requirement

- It better contains all the monomials from which a syntactic factorization may be derived.

$$f = g \odot h \Rightarrow \text{mons}(g) \in \mathcal{M}$$

- It better satisfies condition (**??**)

$$\forall m_i, m_j \in \mathcal{M}, i \neq j \Rightarrow m_i \nmid m_j$$

such that recursive computation of partial syntactic factorizations of all $g_i's$ can be avoided.

About base monomial set (2/2)

Options:

- $\mathcal{M} =$ the set of all the variables X.
- the set of minimal elements of the pairwise gcd set G,

$$G = \{\gcd(m_i, m_j) \mid m_i, m_j \in \text{mons}(f), i \neq j\},$$

for the divisibility relation.

Recursive Calls

$$f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus (g_3 \odot h_3) \oplus r$$

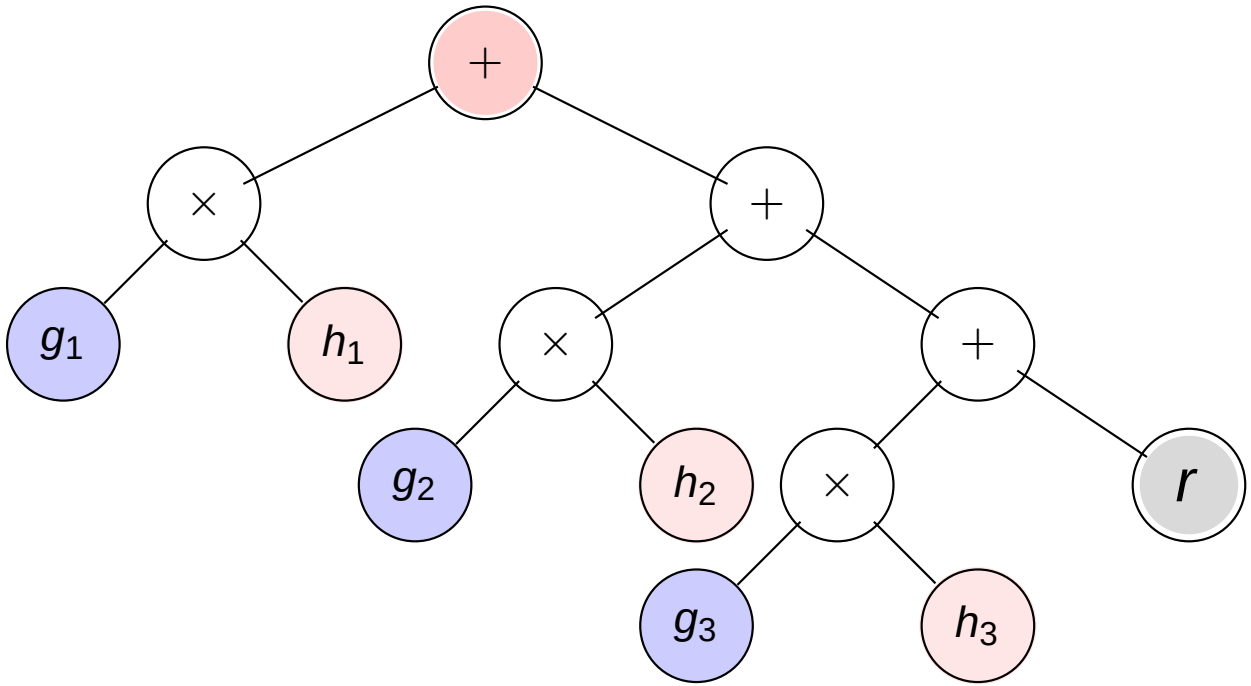


Figure: recursive calls

Syntactic Decomposition

Let T be a binary tree whose internal nodes are the operators $+, -, \times$ and whose leaves belong to $\mathbb{K} \cup X$. Let p_T be the polynomial represented by T. We say that T is a *syntactic decomposition* of p_T if either (1), (2) or (3) holds:

(1) T consists of a single node which is p_T.

(2) if T has root $+$ (resp. $-$) with left subtree T_ℓ and right subtree T_r then we have:
   (a) T_ℓ, T_r are syntactic decompositions of two polynomials $p_{T_\ell}, p_{T_r} \in \mathbb{K}[X]$,
   (b) $p_T = p_{T_\ell} \oplus p_{T_r}$ (resp. $p_T = p_{T_\ell} \ominus p_{T_r}$) holds,

(3) if T has root $\times$, with left subtree T_ℓ and right subtree T_r then we have:
   (a) T_ℓ, T_r are syntactic decompositions of two polynomials $p_{T_\ell}, p_{T_r} \in \mathbb{K}[X]$,
   (b) $p_T = p_{T_\ell} \odot p_{T_r}$ holds.

Complexity

Suppose that f is a polynomial of t terms with total degree d in $\mathbb{K}[x_1, x_2, \ldots, x_n]$. Assume that the input polynomial is given in distributed representation and that each exponent in a monomial is encoded by a machine word.

- A partial syntactic factorization of f can be computed in $O(|\mathcal{M}|^3 t^3 n \log(|\mathcal{M}|t))$ bit operations and $O(|\mathcal{M}|^3 t^3)$ operations in $\mathbb{K}$.
- If $\mathcal{M}$ is chosen to be all the variables, then a syntactic decomposition of f can be computed in $t^3 n^4 d \log(tn)$ bit operations and $t^3 n^3 d$ operations in $\mathbb{K}$.
- If $\mathcal{M}$ is chosen to be the set of minimal elements of pairwise gcd's, then a syntactic decomposition of f can be computed in $O(t^9 nd \log t)$ bit operations and $O(t^9 d)$ operations in $\mathbb{K}$.

Return to Main Results

Computation of parallel evaluation schedule

Question

How to compute a parallel evaluation schedule from a syntactic decomposition of the given polynomial?

Objective

A syntactic decomposition is a binary tree which becomes a directed acyclic graph (DAG) after eliminating common subexpressions. Our objective is to decompose a DAG into p sub-DAGs for a given parameter p, the number of available processors.

- the evaluation of one sub-DAG does not depend on the evaluation of the other.
- these sub-DAGs are balanced in size such that the "span" of the intended parallel evaluation is minimized.

Two steps of DAG evaluation

We first collect all common subexpressions, and treat them as leaves. Then the problem of scheduling DAGs reduces to that of scheduling binary trees, where broadcasting common subexpressions may be necessary.

Example

The DAG on the left represents the polynomial $abc(d+e)+f(abc+h)$.

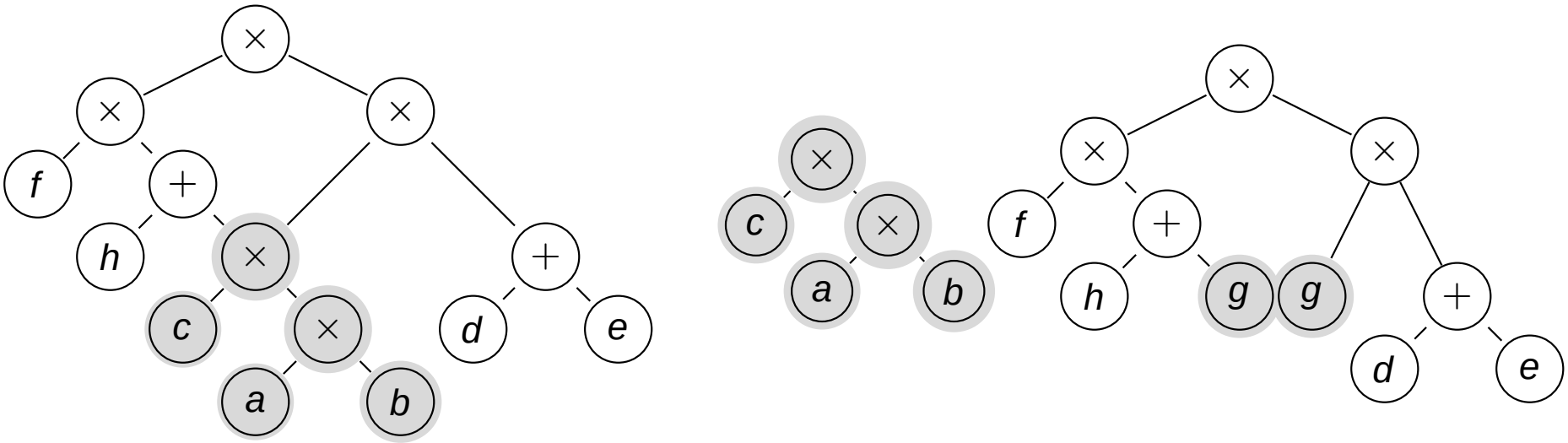


Figure: Evaluate a DAG in two steps

Scheduling of binary trees

By choosing a cut-off value $K > 0$, with a tree traversal, one can find the set M of the maximal subtrees of T such that each subtree has at most K nodes.

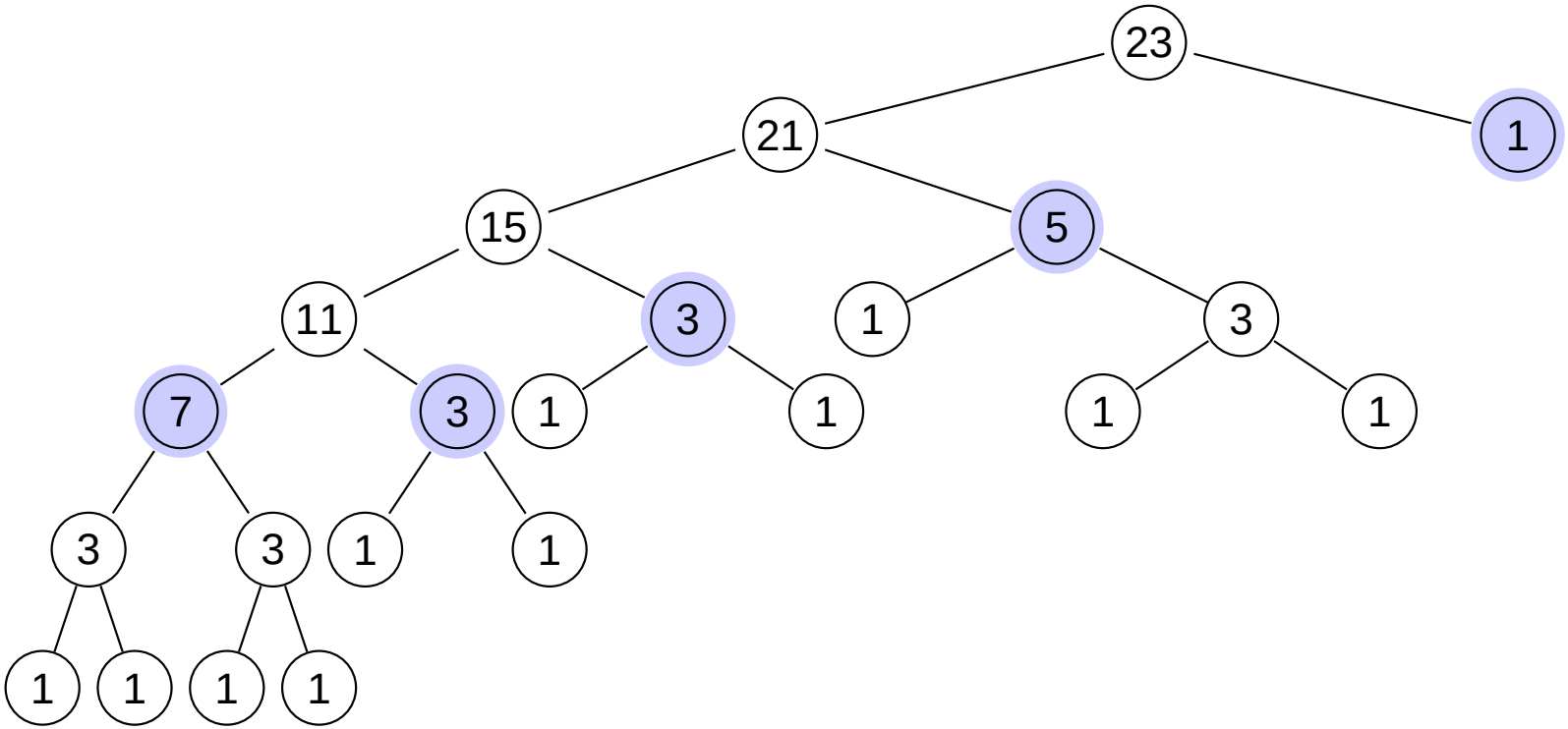


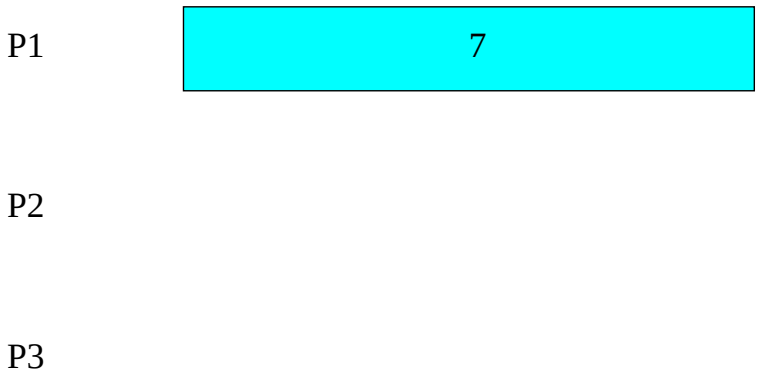
Figure: set p = 3, M = {7, 3, 3, 5, 1}, K = $\lceil \frac{23}{3} \rceil$

Scheduling of binary trees

Multiprocessor scheduling of M is an NP-Complete problem. However, the simple LPT-algorithm[1] (Longest Processing Time) achieves a partition within a factor $4/3$ of optimal.

Example

$$M = \{7, 5, 3, 3, 1\}, p = 3$$



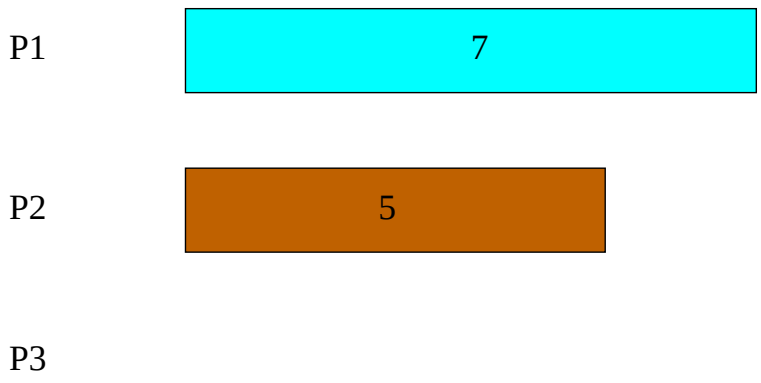
[1] *Bounds on multiprocessing anomalies and related packing algorithms*, R. L. Graham, 1972.

Scheduling of binary trees

Multiprocessor scheduling of M is an NP-Complete problem. However, the simple LPT-algorithm (Longest Processing Time) achieves a partition within a factor $4/3$ of optimal.

Example

$$M = \{7, 5, 3, 3, 1\}, p = 3$$

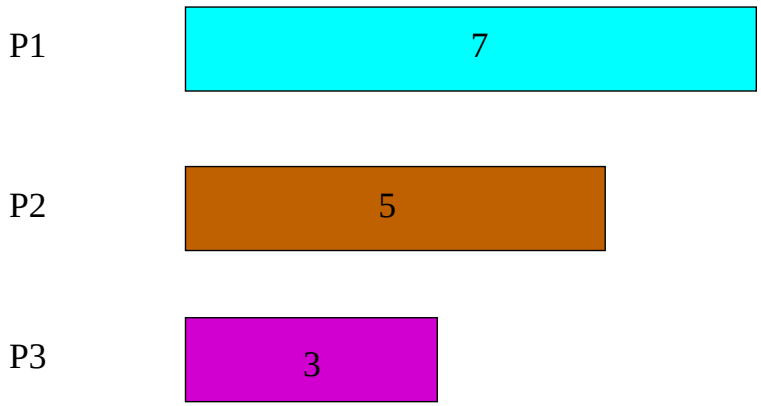


Scheduling of binary trees

Multiprocessor scheduling of M is an NP-Complete problem. However, the simple LPT-algorithm (Longest Processing Time) achieves a partition within a factor $4/3$ of optimal.

Example

$$M = \{7, 5, 3, 3, 1\}, p = 3$$

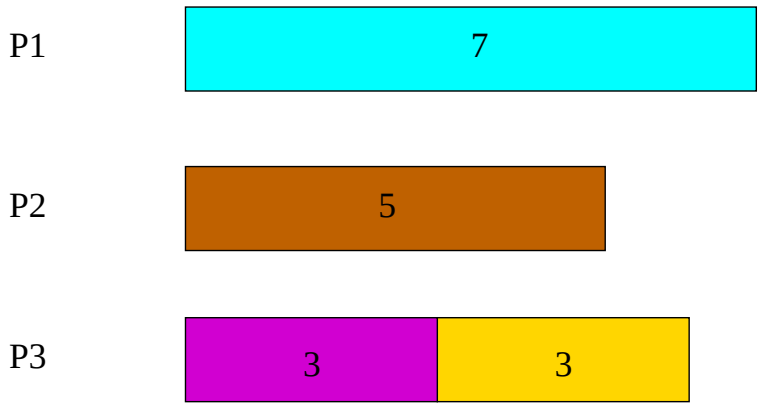


Scheduling of binary trees

Multiprocessor scheduling of M is an NP-Complete problem. However, the simple LPT-algorithm (Longest Processing Time) achieves a partition within a factor $4/3$ of optimal.

Example

$$M = \{7, 5, 3, 3, 1\}, p = 3$$

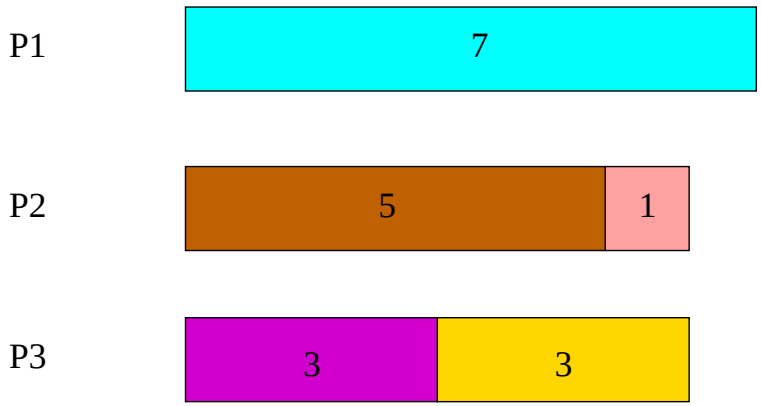


Scheduling of binary trees

Multiprocessor scheduling of M is an NP-Complete problem. However, the simple LPT-algorithm (Longest Processing Time) achieves a partition within a factor $4/3$ of optimal.

Example

$$M = \{7, 5, 3, 3, 1\}, p = 3$$



Scheduling of binary trees

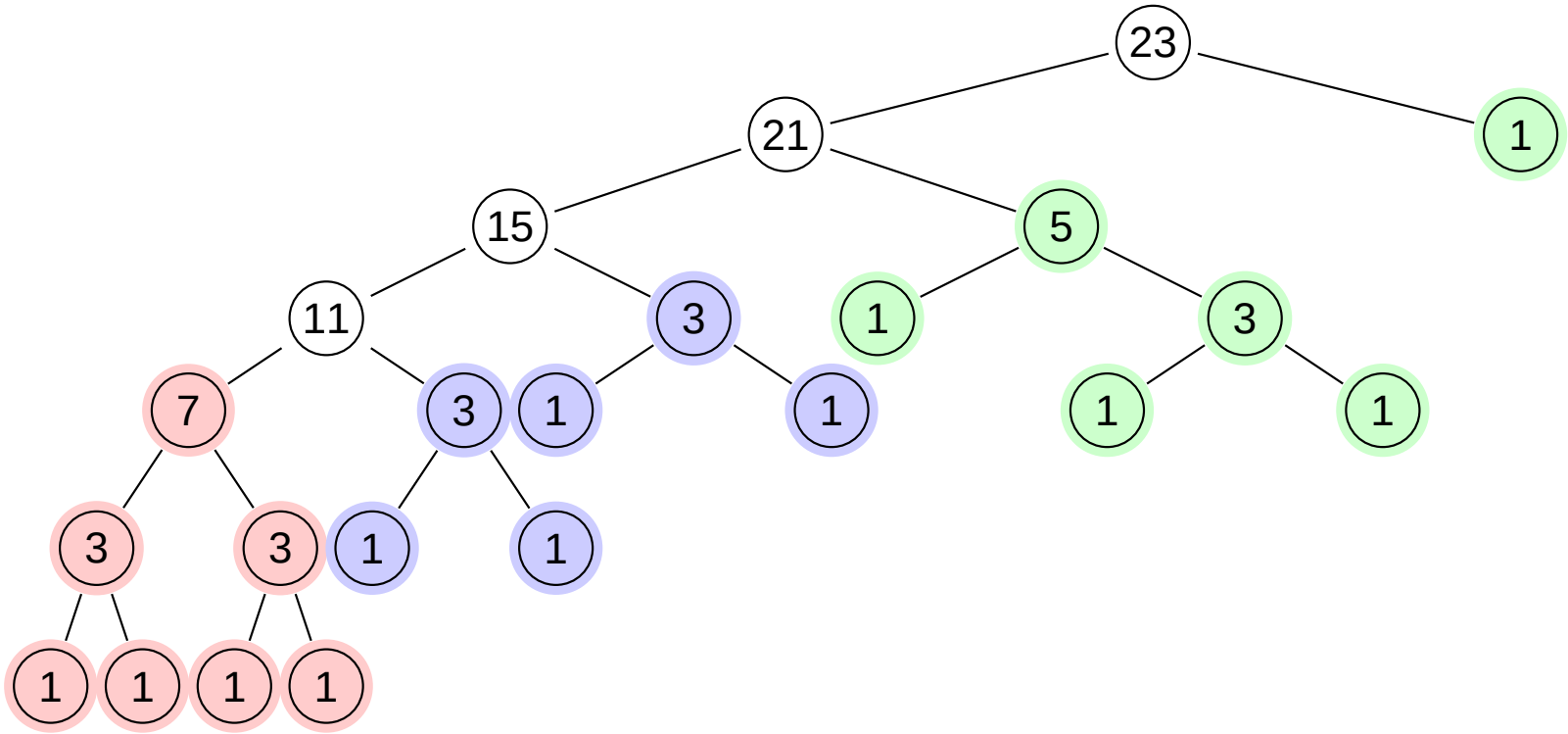


Figure: A 3-schedule of a binary tree

Our test case

- Resultants provide conditions for a given system to have solutions.

linear systems	determinants
non-linear systems	resultants

- Let $a_m, \ldots, a_1, a_0, b_n, \ldots, b_1, b_0$ be independent symbols

$$a = a_m x^m + \cdots + a_1 x + a_0$$

$$b = b_n x^n + \cdots + b_1 x + b_0$$

the number of monomials of resultant(a, b, x) grows exponentially in n and m.

m	5	6	6	7	7	7	8	8	8	8
n	5	5	6	5	6	7	5	6	7	8
#Mon	1696	4605	14869	11380	43166	145330	25917	114080	441145	1524326

Number of operations for evaluating generic resultants

m and n: degree of two polynomials.
Input : direct evaluation of the polynomial.
Horner : Maple's multivariate Horner's rule.
Tryhard : Maple's optimize function with 'tryhard' option.
SD : our hypergraph method. SD + CSE : applying CSE to our output.

m	n	#Mon	Input	Horner	tryhard	SD	SD + CSE
5	5	1696	18185	7779	4056	7134	3543
6	4	1233	13221	6539	3230	4853	2547
6	5	4605	54269	22779	10678	18861	8432
6	6	14869	190890	69909	31760	63492	24701
7	4	2562	30438	14948	6707	9862	4905
7	5	11380	146988	61399	27363	45546	19148
7	6	43166	601633	219341	-	179870	65770
7	7	145330	2166653	697743	-	627584	206840
8	4	4970	63731	19547	12191	18730	8826
8	5	25917	359487	106800	-	101327	39816
8	6	114080	1700662	498410	-	464593	157312
8	7	441145	7028510	2042037	-	1863653	565020
8	8	1524326	25838829	*	-	6648972	1844464

* means that the computation is killed due to 0% CPU usage and 90% memory usage.

- means that the computation does not terminate after 5 days.

Timing in seconds

m and n: degree of two polynomials.
Input : direct evaluation of the polynomial.
Horner : Maple's multivariate Horner's rule.
Tryhard : Maple's optimize function with 'tryhard' option.
SD : our hypergraph method. SD + CSE : applying CSE to our output.

m	n	#Mon	Horner	tryhard	SD
5	5	1696	1.276	868.118	0.617
6	4	1233	0.988	363.970	0.409
6	5	4605	4.868	8658.037	4.820
6	6	14869	24.378	145602.915	43.764
7	4	2562	4.377	1459.343	2.407
7	5	11380	24.305	98225.730	33.156
7	6	43166	108.035	>5 days	404.708
7	7	145330	191.184	>5 days	4252.534
8	4	4970	3.744	6528.992	8.497
8	5	25917	23.858	>5 days	189.259
8	6	114080	145.385	>5 days	3240.737
8	7	441145	930.966	>5 days	45380.056
8	8	1524326	*	>5 days	494362.097

* means that the computation is killed due to 0% CPU usage and 90% memory usage.

Timing in seconds to evaluate different representations

m and n: degree of two polynomials
Input : the input polynomial
SD : syntactic decomposition
SD+CSE : CSE technique applied syntactic decomposition
4-schedule : our computed 4-schedule

m	n	Input	SD	SD+CSE	4-schedule
6	5	144.838	26.681	18.103	9.343
6	6	577.624	185.883	42.788	28.716
7	5	461.981	114.026	40.526	19.560
7	6	1902.813	545.569	138.656	81.270

Figure: Timing to evaluate large polynomials at 100K Points

Return to Main Results

Parallelization of the hypergraph method

Challenge

The algorithm to compute partial syntactic factorization is inherently sequential.

Strategy

We extract parallelism from all subroutines of the computation of partial syntactic factorization, including

- the computation of base monomial set,
- the construction of hypergraph,
- the computation of the largest intersection of hyperedges,
- the computation of the hyperedge with maximal cardinality,
- the product set of two monomial sets,
- the updating process of the hypergraph.

Parallel computation of the base monomial set[2]

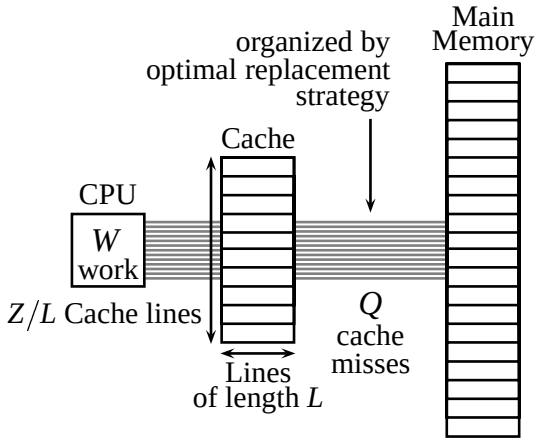
- Our base monomial set is chosen to be the set of minimal elements of the pairwise gcd set G,

$$G = \{\gcd(m_i, m_j) \mid m_i, m_j \in \text{mons}(f), i \neq j\},$$

  where the partial order is chosen to be their divisibility. We call this base monomial set Support of mons(f).
- By Split(A), we mean a partition A^-, A^+ of A such that $|A^-|$ and $|A^+|$ differ at most by 1.
- Union(A, B) accepts two disjoint sets A, B and returns C where $C = A \cup B$;
- A Spawn tells the compiler that the function may run in parallel with the caller.
- A Sync statement indicates that the current function can resume its execution once all functions spawned in its body have completed.

[2] originally proposed in *Parallel Computation of the Minimal Elements of a Poset* (Charles E. Leiserson, Liyun Li, Marc Moreno Maza and Yuzhen Xie, PASCO 2010)

Ideal-Cache Model

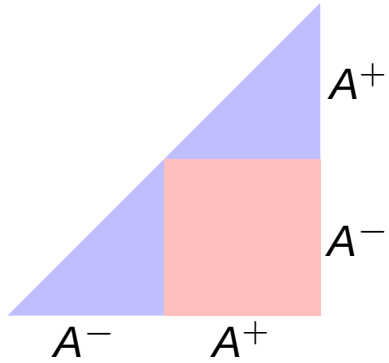


- a computer has a two-level memory hierarchy:
  - an ideal (data) cache of Z words
  - an arbitrarily large main memory
- the cache is partitioned into cache lines, each consisting L consecutive words which are always moved together.

Parallel computation of the base monomial set

Algorithm 1: ParallelSupport

if $|A| \leq$ SupportBase **then**
  return SerialSupport(A);
else
  $(A^-, A^+) \leftarrow$ Split(A);
  $B \leftarrow$ **spawn** SelfSupport(A^-, A^+);
  $C \leftarrow$ **spawn** CrossSupport(A^-, A^+);
  sync;
  $(D_1, D_2) \leftarrow$ ParallelMerge(B, C);
  return Union(D_1, D_2);



$$|A| = n$$

$$\begin{cases} W(n) = & O(n^4) \\ S(n) = & O(n^2) \\ Q(n) = & O(\frac{n^4}{ZL} + \frac{n^2}{L}) \end{cases}$$

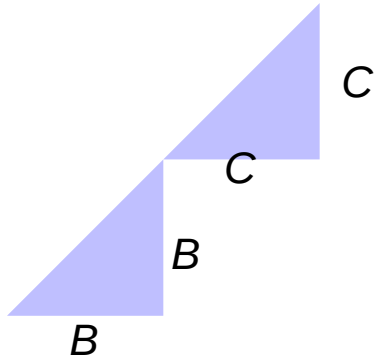
SupportBase should be

- large enough to reduce parallelization overheads
- small enough to increase data locality.

Parallel computation of the base monomial set

Algorithm 2: SelfSupport

$E \leftarrow$ **spawn** ParallelSupport(B);
$F \leftarrow$ **spawn** ParallelSupport(C);
sync;
$(D_1, D_2) \leftarrow$ ParallelMerge(E, F);
return Union(D_1, D_2);



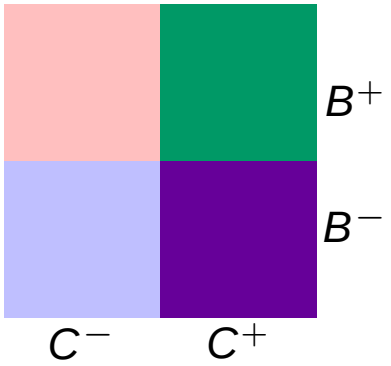
$$|B| = |C| = n$$

$$\begin{cases} W(n) = & O(n^4) \\ S(n) = & O(n^2) \\ Q(n) = & O(\frac{n^4}{ZL} + \frac{n^2}{L}) \end{cases}$$

Parallel computation of the base monomial set

Algorithm 3: CrossSupport

if $|B| \leq$ MergeBase **then**
  return SerialCrossSupport(B, C);
else
  $(B^-, B^+) \leftarrow$ Split(B);
  $(C^-, C^+) \leftarrow$ Split(C);
  $D_1 \leftarrow$ **spawn** CrossSupport(B^-, C^-);
  $D_2 \leftarrow$ **spawn** CrossSupport(B^+, C^-);
  $D_3 \leftarrow$ **spawn** CrossSupport(B^-, C^+);
  $D_4 \leftarrow$ **spawn** CrossSupport(B^+, C^+);
  sync;
  $(E_1, E_2) \leftarrow$ **spawn** ParallelMerge(D_1, D_2);
  $(E_3, E_4) \leftarrow$ **spawn** ParallelMerge(D_3, D_4);
  sync;
  $(F_1, F_2) \leftarrow$
  ParallelMerge(Union(E_1, E_2), Union(E_3, E_4));
  return Union(F_1, F_2);



$$|B| = |C| = n$$

$$\begin{cases} W(n) = & O(n^4) \\ S(n) = & O(n^2) \\ Q(n) = & O(\frac{n^4}{ZL} + \frac{n^2}{L}) \end{cases}$$

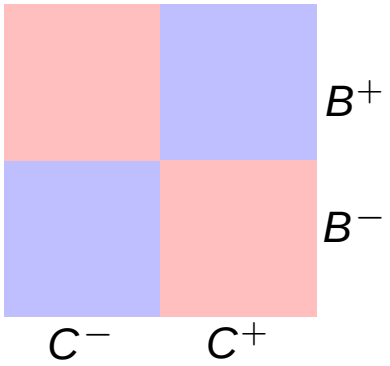
ParallelMerge algorithm

Algorithm 4: ParallelMerge(B, C)

if $|B| \leq$ MergeBase *and* $|C| \leq$ MergeBase **then**
  return SerialMerge(B, C);

else if $|B| >$ MergeBase *and* $|C| >$ MergeBase **then**
  $(B^-, B^+) \leftarrow$ Split(B);
  $(C^-, C^+) \leftarrow$ Split(C);
  $(B^-, C^-) \leftarrow$ **spawn** ParallelMerge(B^-, C^-);
  $(B^+, C^+) \leftarrow$ **spawn** ParallelMerge(B^+, C^+);
  sync;
  $(B^-, C^+) \leftarrow$ **spawn** ParallelMerge(B^-, C^+);
  $(B^+, C^-) \leftarrow$ **spawn** ParallelMerge(B^+, C^-);
  sync;
  return (Union(B^-, B^+), Union(C^-, C^+));

. . . . . .



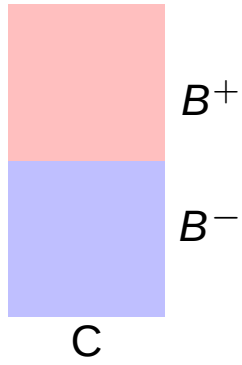
$$|B| = |C| = n$$

$$\begin{cases} W(n) = & O(n^2) \\ S(n) = & O(n) \\ Q(n) = & O(\frac{n^2}{ZL}) \end{cases}$$

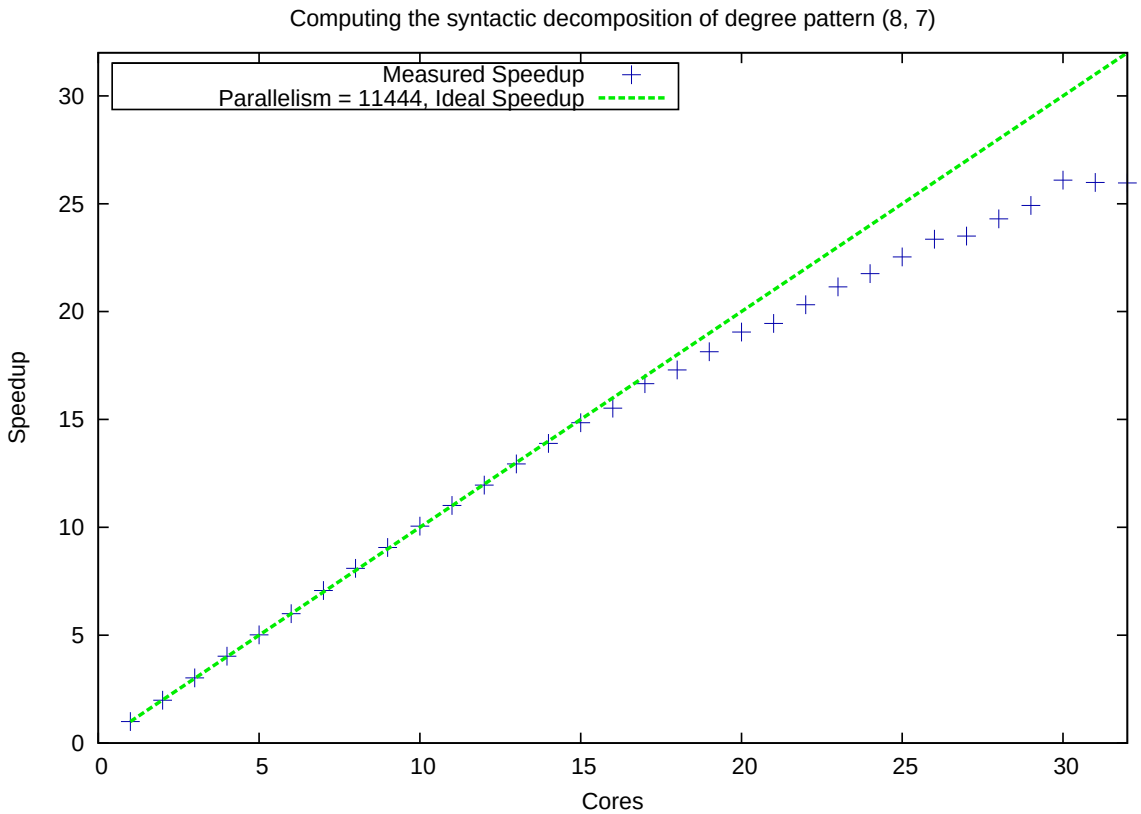
ParallelMerge algorithm

Algorithm 5: ParallelMerge(B, C)

if $|B| \leq$ MergeBase *and* $|C| \leq$ MergeBase **then**
  return SerialMerge(B, C);
else if $|B| >$ MergeBase *and* $|C| >$ MergeBase **then**
  . . . . . .
else if $|B| >$ MergeBase *and* $|C| \leq$ MergeBase **then**
  $(B^-, B^+) \leftarrow$ Split(B);
  $(B^-, C) \leftarrow$ ParallelMerge(B^-, C);
  $(B^+, C) \leftarrow$ ParallelMerge(B^+, C);
  return (Union$(B^-, B^+), C)$;
. . . . . .



Scalability Analysis by Cilkview



Syntactic decomposition computation of the resultant of generic polynomials with degree 8 and 7.

Conclusion

- Given a polynomial in expanded form, we propose an algorithm to compute a syntactic decomposition of it which permits a cheaper evaluation.
- Our computation runs in polynomial time with the degree, number of variables and number of terms of the input polynomial.
- We have implemented our algorithm in the Cilk++ concurrency platform and our implementation achieves near linear speedup on 16 cores with large enough input.
- We have introduced an algorithm to generate a parallel evaluation schedule of a syntactic decomposition. It has been realized to produce a Cilk++ program that is ready to be compiled.
- For some large polynomial, our resulting schedule provides a 10-times-faster evaluation comparing to the direct evaluation of its straight-line program (SLP) representation (both conducted sequentially).

Acknowledgments

- I am heartily thankful to my supervisor, Marc Moreno Maza. This thesis would not have been possible without his support from the initial to the final level.
- Great thanks to my colleagues Yuzhen Xie, Xin Li, Changbo Chen, Wei Pan, Sardar Anisul Haque, Paul Vrbik, Rong Xiao for providing a stimulating and fun environment in which to learn and grow.
- I am thankful to the examination committee, David Jeffrey, Hanan Lutfiyya and Stephen Watt and the chair, Eric Schost for their precious time.
- I was partially supported by the Shared Hierarchical Academic Research Computing Network (SHARCNET) and the MITACS full project *Mathematics of Computer Algebra and Analysis* (MOCAA).
- Many thanks to others who have contributed in assorted ways to my project.

Serial base monomial set algorithm

Algorithm 6: SerialSupport

$\mathcal{M} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** t **do**
  for $j \leftarrow i + 1$ **to** t **do**
    $g \leftarrow \text{Gcd}(M_i, M_j)$; $is\_min \leftarrow true$;
    for $m \in \mathcal{M}$ **do**
      if $m \mid g$ **then**
        $is\_min \leftarrow false$;
        break;
      else if $g \mid m$ **then**
        $\mathcal{M} \leftarrow \mathcal{M} \setminus \{m\}$;
    if $is\_min = true$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{g\}$;
return $\mathcal{M}$;

Serial cross monomial set algorithm

Algorithm 7: SerialCrossSupport

$\mathcal{M} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** $|B|$ **do**
  for $j \leftarrow 1$ **to** $|C|$ **do**
    $g \leftarrow \mathrm{Gcd}(B_i, C_j)$;
    $is\_min \leftarrow true$;
    for $m \in \mathcal{M}$ **do**
      if $m \mid g$ **then**
        $is\_min \leftarrow false$;
        break;
      else if $g \mid m$ **then**
        $\mathcal{M} \leftarrow \mathcal{M} \setminus \{m\}$;
    if $is\_min = true$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{g\}$;
return $\mathcal{M}$;

Serial merge algorithm

Algorithm 8: SerialMerge

```
if |B| = 0 or |C| = 0 then
    return (B, C);
else
    for i ← 1 to |B| do
        for j ← 1 to |C| do
            if c_j is unmarked then
                if c_j ⪯ b_i then
                    Merge(b_i, c_j), Mark b_i and break inner loop;
                if b_i ⪯ c_j then
                    Merge(c_j, b_i), Mark c_j;
    B ← {unmarked elements in B};
    C ← {unmarked elements in C};
    return (B, C);
```

Why syntactic?

Suppose that f admits a syntactic factorization $g\,h$ while nothing is known about g and h, except their numbers of terms. Then, one can set up a system of polynomial equations to compute the terms of g and h. For instance with $t_f = 4$ and $t_g = t_h = 2$, let

$$f = M + N + P + Q, g = X + Y, h = Z + T$$

Up to renaming the terms of f, the following system must have a solution:

$$XZ = M,\ XT = P,\ YZ = N \quad \text{and} \quad YT = Q.$$

This implies that $M/P = N/Q$ holds. Then, one can check that

$$\left(g, g', \frac{M}{g}, \frac{N}{g'}\right)$$

is a solution for (X, Y, Z, T), where $g = \gcd(M, P)$ and $g' = \gcd(N, Q)$.

Functional decomposition

Let $f(\vec{x}), h_1(\vec{x}), \ldots, h_d(\vec{x})$ and $g(\vec{x})$ be polynomials over K. If

$$f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_d(\vec{x})),$$

then we call $g, h_1, , \ldots, h_d$ a functional decomposition of f.

- For univariate polynomials in "tame" case, the time bound given by von zur Gathen[3] is $O(n \log^2 n \log\log n)$, $O(n \log^2 n)$ if F supports an FFT.
- Given a monic n variable polynomial $f \in \mathbb{K}[X]$ of total degree d and and $r \in \mathbb{N}$, Dickerson[4] shows how to find a monic $g \in \mathbb{K}[X]$ of total degree r and monic $h \in \mathbb{K}[X]$ of degree $s = d/r$ such that $f = g \circ h$. The computation requires $O(d^{3n})$ field operations.

[3] *Functional decomposition of polynomials: the tame case.* J. Symb. Comput., 1990.

[4] *Polynomial Decomposition Algorithms for Multivariate Polynomial*, Cornell University, Technical Report 87-826, 1987.

Polynomial factorization

- A nonconstant polynomial $f \in \mathbb{F}_q[x]$, where q is an odd prime power can be reduced to monic irreducible factors and their multiplicities in $O\tilde{}(n^2 \log q)$ arithmetic operations in $\mathbb{F}_q$ [5] .
- A non-zero polynomial $f \in \mathbb{Q}[x]$ of degree $n \geq 1$ can be factorized into irreducibie factors in $O\left(n^{12} + n^9(\log(|f|))^3\right)$ bit operations [6], where

$$|\sum_i a_i x^i| = (\sum_i a_i^2)^{1/2}.$$

[5] *Modern Computer Algebra.*

[6] *Factoring polynomials with rational coefficients*, A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, 1982.

Coefficients solving of candidate syntactic factorization

Proposition

Let $F_1, F_2, \ldots, F_c$ be the monomials and $f_1, f_2, \ldots, f_c$ be the coefficients of a polynomial $f \in \mathbb{K}[X]$, such that $f = \sum_{i=1}^{c} f_i F_i$. Let $a, b > 0$ be two integers such that $c = ab$. Suppose we are given two lists monomials $G = \{G_1, G_2, \ldots, G_a\}$ and $H = \{H_1, H_2, \ldots, H_b\}$ such that the products $G_i H_j$ are all in $\mathrm{mons}(f)$ *and are pairwise different. Then, within $O(c)$ operations in $\mathbb{K}$ and $O(nc \log c)$ bit operations, one can decide whether $f = g \odot h$,* $\mathrm{mons}(g) = G$ *and* $\mathrm{mons}(h) = H$ *all hold. Moreover, if such a syntactic factorization exists it can be computed within the same time bound.*

Code generation challenge

The following figure illustrates the typical sizes (the generic resultant with $m = n = 6$) of the source files, object files and executable files generated from the four methods. The first data row shows the number of lines in each SLP.

	Input	SD	SD+CSE	4-schedule
#line	319091	121463	24820	24820
src	16 MB	6MB	1MB	1MB
obj	22 MB	9MB	2MB	2MB
exe	16 MB	6MB	2MB	2MB

Figure: file sizes of different methods

The gcc 4.2.4 compiler takes 3-4 minutes to compile a file of size 1MB and more than 4 hours to compile a file of size 12MB. For such large file, the memory usage goes up to 100%. We have to divide a large file into many small files in order to compile it.

Evaluation schedule of generic resultants

m and n: degree of two polynomials
T : the number of operations in syntactic decomposition
#CS : the number of operations in common subexpressions
T' : the number of operations got scheduled

m	n	T	$\#CS$	T'	4-schedule
6	5	8510	1455	7030	1760, 1761, 1755, 1754
6	6	24820	4491	20294	5082, 5069, 5072, 5071
7	5	19293	3169	16073	4029, 4012, 4017, 4015
7	6	66022	11167	54792	13694, 13699, 13717, 13682
7	7	207289	35096	172073	43195, 42981, 42949, 42948
8	5	40051	6812	33186	8305, 8287, 8292, 8302
8	6	157784	28461	129217	32347, 32289, 32281, 32300
8	7	565909	103311	462395	115625, 115589, 115603, 115578
8	8	1846280	345446	1500295	375772, 374969, 374779, 374775

Figure: Parallel evaluation 4-schedule