Memory-efficient polynomial arithmetic

Pascal Giorgi¹  Bruno Grenet¹  Daniel S. Roche²
LIP, ÉNS de Lyon — 11 apr. 2019

¹ LIRMM, Université de Montpellier

² CS Department, US Naval Academy
Polynomial arithmetic

- Multiplication: \( M(n) \)
  - Naïve: \( 2n^2 + 2n - 1 \)
  - Karatsuba: \( < 6.5n^{\log_2 3} \)
  - Toom-3: \( < 18.75n^{\log_3 5} \)
  - FFT-based: \( 4.5n \log n + O(n) \) or \( O(n \log n \log \log n) \)
Polynomial arithmetic

- **Multiplication**: $M(n)$
  - Naïve: $2n^2 + 2n - 1$
  - Karatsuba: $< 6.5n^{\log_2 3}$
  - Toom-3: $< 18.75n^{\log_3 5}$
  - FFT-based: $4.5n \log n + O(n)$ or $O(n \log n \log \log n)$

- **Other tasks**:
  - Euclidean division: $5M(n) + o(M(n))$
  - GCD: $O(M(n) \log n)$
  - Evaluation & interpolation: $O(M(n) \log n)$
  - Power series computations: $O(M(n))$ or $O(M(n) \log n)$
  - ...
Polynomial arithmetic

- Multiplication: $M(n)$
  - Naïve: $2n^2 + 2n - 1$
  - Karatsuba: $< 6.5n^{\log_2 3}$
  - Toom-3: $< 18.75n^{\log_3 5}$
  - FFT-based: $4.5n \log n + O(n)$ or $O(n \log n \log \log n)$

- Other tasks:
  - Euclidean division: $5M(n) + o(M(n))$
  - GCD: $O(M(n) \log n)$
  - Evaluation & interpolation: $O(M(n) \log n)$
  - Power series computations: $O(M(n))$ or $O(M(n) \log n)$
  - ...
Space complexity of polynomial arithmetic

- Quadratic multiplication algorithm: $O(1)^{1}$
- Karatsuba, Toom-3, FFT: $O(n)$
- Other tasks: often $O(n)$

1. Models to be defined later.
Space complexity of polynomial arithmetic

- Quadratic multiplication algorithm: $O(1)^1$
- Karatsuba, Toom-3, FFT: $O(n)$
- Other tasks: often $O(n)$

- Improvements on Karatsuba’s algorithm:
  - Thomé (2002): $n + O(\log n)$
  - Roche (2009): $O(\log n)$

  $\rightarrow$ time complexity multiplied by a constant

1. Models to be defined later.
Space complexity of polynomial arithmetic

- Quadratic multiplication algorithm: $O(1)$\(^1\)
- Karatsuba, Toom-3, FFT: $O(n)$
- Other tasks: often $O(n)$

- Improvements on Karatsuba’s algorithm:
  - Thomé (2002): $n + O(\log n)$
  - Roche (2009): $O(\log n)$
  $\rightarrow$ time complexity multiplied by a constant

- Improvements on FFT-based algorithms:
  - Roche (2009): $O(1)$ if $n = 2^k$
  - Harvey & Roche (2010): $O(1)$
  $\rightarrow$ time complexity multiplied by a constant

---

1. Models to be defined later.
Space-complexity models

Algebraic-RAM machine:

→ *Standard* registers of size $O(\log n)$
→ *Algebraic* registers containing one coefficient
Space-complexity models

*Algebraic*-RAM machine:
   → *Standard* registers of size $O(\log n)$
   → *Algebraic* registers containing one coefficient

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication

Thomé (2002), Roche (2009) and Harvey & Roche (2010)

Reasonable from a programmer's viewpoint

Read-write input and output

- Too permissive in general
  - Variant: inputs must be restored at the end
Space-complexity models

**Algebraic-RAM machine:**

→ *Standard* registers of size $O(\log n)$  
→ *Algebraic* registers containing one coefficient

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication

- Read-only input / read-write output
  - Thomé (2002), Roche (2009) and Harvey & Roche (2010)
  - *Reasonable* from a programmer’s viewpoint
Space-complexity models

Algebraic-RAM machine:
  \( \rightarrow \) Standard registers of size \( O(\log n) \)
  \( \rightarrow \) Algebraic registers containing one coefficient

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound \( \Omega(n^2) \) on time \( \times \) space for multiplication

- Read-only input / read-write output
  - Thomé (2002), Roche (2009) and Harvey & Roche (2010)
  - Reasonable from a programmer’s viewpoint

- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end
Space-complexity models

**Algebraic-RAM machine:**

- Standard registers of size $O(\log n)$
- Algebraic registers containing one coefficient

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication

- Read-only input / read-write output
  - Thomé (2002), Roche (2009) and Harvey & Roche (2010)
  - *Reasonable* from a programmer’s viewpoint

- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end
Previous results

- Karatsuba’s algorithm:
  - Divide-and-Conquer: \((f_0 + X^{\frac{n}{2}} f_1) \cdot (g_0 + X^{\frac{n}{2}} g_1)\)
    
    \[
    = f_0g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0g_0 - f_1g_1)X^{\frac{n}{2}} + f_1g_1X^n
    \]
Previous results

- Karatsuba’s algorithm:
  - Divide-and-Conquer: \((f_0 + \frac{n}{2} f_1) \cdot (g_0 + \frac{n}{2} g_1)\)
    \[= f_0 g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1)X^{\frac{n}{2}} + f_1 g_1 X^n\]

- Thomé’02: Careful use of \(n\) temp. registers + \(O(\log n)\) stack

- Roche’09: *half-additive* version \(\Rightarrow\) only \(O(\log n)\) stack
  \((h_\ell \leftarrow h_\ell + fg\text{ where }\deg(h_\ell) < \deg(f), \deg(g))\)

- FFT-based algorithms:
  - \((F, G) \rightarrow (F(\omega^i), G(\omega^i))\)
    \[\rightarrow FG(\omega^i)\]
  - Every \(\rightarrow\) is in-place (overwriting) but \# points is \(1 + \deg(FG)\)
  \[\Rightarrow size((F(\omega^i), G(\omega^i))) = 2 \times size(FG)\]

- Roche’09: Compute half of the result + recurse

- Harvey-Roche’10: same with TFT (vdH’04)
Previous results

- Karatsuba’s algorithm:
  - Divide-and-Conquer: \((f_0 + X^{\frac{n}{2}} f_1) \cdot (g_0 + X^{\frac{n}{2}} g_1)\)
    \[= f_0 g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1)X^{\frac{n}{2}} + f_1 g_1 X^n\]
  - Thomé’02: Careful use of \(n\) temp. registers + \(O(\log n)\) stack
  - Roche’09: half-additive version \(\Rightarrow\) only \(O(\log n)\) stack
    \((h_\ell \leftarrow h_\ell + fg\) where \(\deg(h_\ell) < \deg(f), \deg(g)\))

- FFT-based algorithms:
  - \((F, G) \rightarrow (F(\omega^i), G(\omega^i))_i \rightarrow FG(\omega^i)_i \rightarrow FG\)
Previous results

- Karatsuba’s algorithm:
  - Divide-and-Conquer: 
    
    \[
    (f_0 + X^{\frac{n}{2}} f_1) \cdot (g_0 + X^{\frac{n}{2}} g_1) = f_0 g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1) X^{\frac{n}{2}} + f_1 g_1 X^n
    \]
  - Thomé’02: Careful use of \( n \) temp. registers + \( O(\log n) \) stack
  - Roche’09: halt-additive version \( \rightsquigarrow \) only \( O(\log n) \) stack

- FFT-based algorithms:
  - \((F, G) \rightarrow (F(\omega^i), G(\omega^i))_i \rightarrow FG(\omega^i)_i \rightarrow FG\)
  - Every \( \rightarrow \) is in-place (overwriting) but \# points is \( 1 + \deg(FG) \)
  - \( \rightsquigarrow \) \( \text{size}((F(\omega^i), G(\omega^i))_i) = 2 \text{size}(FG) \)
Previous results

- **Karatsuba’s algorithm:**
  - Divide-and-Conquer: \( (f_0 + \frac{n}{2} f_1) \cdot (g_0 + \frac{n}{2} g_1) \)
    \[
    = f_0g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0g_0 - f_1g_1)X^{n/2} + f_1g_1X^n
    \]
  - Thomé’02: Careful use of \( n \) temp. registers + \( O(\log n) \) stack
  - Roche’09: *half-additive* version ⇝ only \( O(\log n) \) stack
    \( h_\ell \leftarrow h_\ell + fg \) where \( \deg(h_\ell) < \deg(f), \deg(g) \)

- **FFT-based algorithms:**
  - \( (F, G) \rightarrow (F(\omega^i), G(\omega^i)) \); \( FG(\omega^i) \); \( FG \)
  - Every \( \rightarrow \) is in-place (overwriting) but \# points is \( 1 + \deg(FG) \)
  - \( \rightsquigarrow \) \( \text{size}((F(\omega^i), G(\omega^i))_i) = 2 \text{size}(FG) \)
  - Roche’09: Compute half of the result + recurse
  - Harvey-Roche’10: same with TFT (vdH’04)
Our problematic

Can *every* polynomial multiplication algorithm be performed without extra memory?
Can *every* polynomial multiplication algorithm be performed without extra memory?

- $O(1)$-space Karatsuba’s algorithm?
- What about Toom-Cook algorithm?
Can every polynomial multiplication algorithm be performed without extra memory?

- $O(1)$-space Karatsuba’s algorithm?
- What about Toom-Cook algorithm?
- What about other products (short and middle)?
Can every polynomial multiplication algorithm be performed without extra memory?

- $O(1)$-space Karatsuba’s algorithm?
- What about Toom-Cook algorithm?
- What about other products (short and middle)?

Results:
- Yes!
- Almost (for other products)
Outline

Polynomial products and linear maps

Space-preserving reductions

In-place algorithms from out-of-place algorithms
Polynomial products and linear maps
Short product

\[
\text{Short product} = n \times (n-1)
\]

- Useful in other algorithms
- Time complexity: \( M(n) \)
- Space complexity: \( O(n) \)
Short product

$$n \times n - 1$$

- Low short product: product of truncated power series
- Useful in other algorithms
- Time complexity: $O(n)$
- Space complexity: $O(n)$
Short product

- Low short product: product of truncated power series
- Useful in other algorithms
- Time complexity: $M(n)$
- Space complexity: $O(n)$
Middle product

\[ \times \]

\[ 3n - 2 \]

\[ 2n - 1 \]

Useful for Newton iteration

\[ G \leftarrow G \left( 1 - GF \right) \mod X \]

\[ H \]

\[ \text{Time complexity: } M(n) \rightarrow \text{Tellegen's transposition} \]

\[ \text{Space complexity: } O(n) \]

\[ O(1) \text{ space in the most permissive model via transposition of Harvey-Roche algorithm (Bostan-Lecerf-Schost'03)} \]
Middle product

\[ \text{middle product} = \times n - 1 \]

Useful for Newton iteration

\[ G \leftarrow G \left( 1 - GF \right) \mod X^{2n} \]

division, square root, ...

Time complexity: \( M(n) \)

Space complexity:

\[ O(n) \]

\[ O(1) \]

space in the most permissive model via transposition of Harvey-Roche algorithm (Bostan-Lecerf-Schost'03)
**Middle product**

- Useful for Newton iteration
  - \( G \leftarrow G(1 - GF) \mod X^{2n} \) with \( GF = 1 + X^n H \)
  - division, square root, ... 
- Time complexity: \( M(n) \rightarrow \) Tellegen’s transposition
- Space complexity: \( O(n) \)
- \( O(1) \) space in the most permissive model via transposition of Harvey-Roche algorithm (Bostan-Lecerf-Schost’03)
Multiplications as linear maps

\[ = \times n^2 n - 1 \]
Multiplications as linear maps

\[ n \times n = n - 1 \]
Multiplications as linear maps

\[ \times \quad 3n - 1 \]
Multiplications as linear maps
Multiplications as linear maps

- Full product (FP)
- Short products ($SP_{lo}$ and $SP_{hi}$)
- Middle product (MP)
Space-preserving reductions
Relative difficulties of products

- Without space restrictions:
  - \( SP \leq FP \) and \( FP \leq SP_{lo} + SP_{hi} \)
  - \( MP \equiv FP \) (transposition)
  - \( MP \leq SP_{lo} + SP_{hi} + (n - 1) \) additions
Relative difficulties of products

- Without space restrictions:
  - \( SP \leq FP \) and \( FP \leq SP_{lo} + SP_{hi} \)
  - \( MP \equiv FP \) (transposition)
  - \( MP \leq SP_{lo} + SP_{hi} + (n - 1) \) additions

- Size of inputs and outputs:
  - \( FP : n \times n \rightarrow 2n - 1 \)
  - \( SP_{lo} : n \times n \rightarrow n \)
  - \( SP_{hi} : n - 1 \times n - 1 \rightarrow n - 1 ; \)
  - \( MP : 2n - 1 \times n \rightarrow n \)
Relative difficulties of products

- Without space restrictions:
  - $SP \leq FP$ and $FP \leq SP_{lo} + SP_{hi}$
  - $MP \equiv FP$ (transposition)
  - $MP \leq SP_{lo} + SP_{hi} + (n-1)$ additions

- Size of inputs and outputs:
  - $FP : n \times n \rightarrow 2n - 1$
  - $SP_{lo} : n \times n \rightarrow n$
  - $SP_{hi} : n-1 \times n-1 \rightarrow n-1$
  - $MP : 2n-1 \times n \rightarrow n$

Reductions unusable in space-restricted settings!
A relevant notion of reduction

Definitions

- **TISP**(\(t(n), s(n)\)): computable in time \(t(n)\) and space \(s(n)\)
- **\(A \leq_c B\)**: \(A\) computable with oracle \(B\) and
  - constant number \(c\) of calls to oracle
  - negligible extra time
  - without extra space \((O(1))\)
- **\(A \equiv_c B\)**: \(A \leq_c B\) and \(B \leq_c A\)
A relevant notion of reduction

Definitions
- **TISP**$(t(n), s(n))$: computable in time $t(n)$ and space $s(n)$
- $A \leq_c B$: $A$ computable with oracle $B$ and
  - constant number $c$ of calls to oracle
  - negligible extra time
  - without extra space ($O(1)$)
- $A \equiv_c B$: $A \leq_c B$ and $B \leq_c A$

Proposition
If $B \in \text{TISP}(t(n), s(n))$ and $A \leq_c B$, then

$$A \in \text{TISP}(c t(n) + o(t(n)), s(n) + O(1))$$
Results

Theorem

\[ \text{FP} \leq 2 \leq 1 \equiv 1 \]

\[ \text{SP}_{\text{lo}} \leq 1 \leq \text{MP} \]

\[ \text{SP}_{\text{hi}} \]
- Use of *fake padding* (in input, **not** in output!)
Use of *fake padding* (in input, **not** in output!)

- $SP_{lo}(n) \leq MP(n)$; $SP_{hi}(n) \leq MP(n - 1)$
Use of *fake padding* (in input, **not** in output!)

- $\text{SP}_{\text{lo}}(n) \leq \text{MP}(n)$; $\text{SP}_{\text{hi}}(n) \leq \text{MP}(n - 1)$
- $\text{FP}(n) \leq \text{SP}_{\text{hi}}(n) + \text{SP}_{\text{lo}}(n) \leq \text{MP}(n) + \text{MP}(n - 1)$
Half-additive full product: $h \leftarrow h + f \cdot g$

\[ n - 1 \]

$FP_{lo}^+$:

\[ n \]

\[ f \times g \]

\[ n \]
Half-additive full product: $h \leftarrow h + f \cdot g$
Half-additive full product: \( h \leftarrow h + f \cdot g \)

Remark \( \text{FP}_\text{lo}^+ \equiv_1 \text{FP}_\text{hi}^+ \)

Theorem \( \text{FP}^+ \leq_{3/2} \text{SP} \) and \( \text{SP} \leq_2 \text{FP}^+ \)
From SP to FP

\[ \text{SP} \times \text{lo}(n) \leq \text{SP} \text{lo}(n) + \text{SP} \text{hi}(n) + n - 1 \]
From SP to FP$^+$
From SP to FP⁺

\[ \text{FP} \circ \text{lo}(n) \leq \text{SP}\circ \text{lo}(n) + \text{SP}\circ \text{hi}(n) + n - 1 \]
From SP to FP$^+$

\[ \text{FP}_n \leq \text{SP}_{lo}(n) + \text{SP}_{hi}(n) + n - 1 \]
From SP to FP$^+$

$$\text{FP}^{+}_{lo}(n) \leq \text{SP}_{lo}(n) + \text{SP}_{hi}(n) + n - 1$$
From FP$^+$ to SP

\[
(f_0 + X^{\lfloor n/2 \rfloor} f_1) \cdot (g_0 + X^{\lceil n/2 \rceil} g_1) = f_0 g_0 + X^{\lceil n/2 \rceil} (f_0 g_1 + f_1 g_0) \mod X^n
\]
From \( \text{FP}^+ \) to \( \text{SP} \)

\[
(f_0 + X^{n/2} f_1) \cdot (g_0 + X^{n/2} g_1) = f_0 g_0 + X^{n/2} (f_0 g_1 + f_1 g_0) \mod X^n
\]
From FP\(^+\) to SP

\[
(f_0 + X^{n/2} f_1) \cdot (g_0 + X^{n/2} g_1) = f_0 g_0 + X^{n/2} (f_0 g_1 + f_1 g_0) \mod X^n
\]
From FP$^+$ to SP

$$(f_0 + X^{\lceil n/2 \rceil} f_1) \cdot (g_0 + X^{\lceil n/2 \rceil} g_1) = f_0 g_0 + X^{\lceil n/2 \rceil} (f_0 g_1 + f_1 g_0) \mod X^n$$
From $FP^+$ to $SP$

$\left( f_0 + X^{\lceil n/2 \rceil} f_1 \right) \cdot \left( g_0 + X^{\lceil n/2 \rceil} g_1 \right) = f_0 g_0 + X^{\lceil n/2 \rceil} (f_0 g_1 + f_1 g_0) \mod X^n$
From FP\(^+\) to SP

\[
(f_0 + X^{n/2} f_1) \cdot (g_0 + X^{n/2} g_1) = f_0 g_0 + X^{n/2} (f_0 g_1 + f_1 g_0) \mod X^n
\]
From $FP^+$ to $SP$

\[
(f_0 + X^{\lceil n/2 \rceil} f_1) \cdot (g_0 + X^{\lceil n/2 \rceil} g_1) = f_0g_0 + X^{\lceil n/2 \rceil}(f_0g_1 + f_1g_0) \mod X^n
\]
From FP$^+$ to SP

\[(f_0 + X^{\lceil n/2 \rceil} f_1) \cdot (g_0 + X^{\lceil n/2 \rceil} g_1) = f_0 g_0 + X^{\lceil n/2 \rceil} (f_0 g_1 + f_1 g_0) \mod X^n\]

\[\text{SP}_{lo}(n) \leq \text{FP}(\lfloor n/2 \rfloor) + \text{FP}^+_{lo}(\lfloor n/2 \rfloor) + \text{FP}^+_{hi}(\lceil n/2 \rceil)\]
Converse directions?

- From FP to SP:
  - problem with the output size
  - without space restriction: is $\text{SP}(n) \simeq \text{FP}(n/2)$?
Converse directions?

- **From FP to SP:**
  - problem with the output size
  - without space restriction: is $\text{SP}(n) \simeq \text{FP}(n/2)$?

- **From SP to MP:**
  - partial result:
    - up to $\log(n)$ increase in time complexity
    - techniques from next part
  - without space restriction or in a permissive model
    - FP to MP through Tellegen’s transposition principle
In-place algorithms from out-of-place algorithms
In-place algorithms parametrized by out-of-place algorithm
  - Out-of-place: Uses $cn$ extra space
  - Constant $c$ known to the algorithm
In-place algorithms parametrized by out-of-place algorithm
- Out-of-place: Uses $cn$ extra space
- Constant $c$ known to the algorithm

Goal:
- Space complexity: $O(1)$
- Time complexity: closest to the out-of-place algorithm
In-place algorithms parametrized by out-of-place algorithm
  - Out-of-place: Uses $cn$ extra space
  - Constant $c$ known to the algorithm

Goal:
  - Space complexity: $O(1)$
  - Time complexity: closest to the out-of-place algorithm

Technique:
  - Oracle calls in smaller size
  - Tail recursive call
  - Fake padding
Tail recursion and fake padding

- Tail recursion:
  - Only one recursive call + last (or first) instruction
  - No need of recursive stack $\Rightarrow$ avoid $O(\log n)$ extra space

- Fake padding:
  - Pretend to pad inputs with zeroes
  - Make the data structure responsible for it
  - $O(1)$ increase in memory

Cf. strides in dense linear algebra

OK in inputs, not in outputs!
Tail recursion and fake padding

- **Tail recursion:**
  - Only one recursive call + last (or first) instruction
  - No need of recursive stack \(\leadsto\) avoid \(O(\log n)\) extra space

- **Fake padding:**
  - Pretend to pad inputs with zeroes
  - Make the data structure responsible for it
    - \(O(1)\) increase in memory
    - *Cf.* strides in dense linear algebra
  - OK in inputs, not in outputs!
In-place $FP^+$ from out-of-place $FP$

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$
\[(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}\]
(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}
\[(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}\]
(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}
In-place FP$^+$ from out-of-place FP

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$
\[
k \times \left\lceil \frac{n}{k} \right\rceil - 1 \leq n - k - 1 = k \leq n + 1 \]
\begin{itemize}
  \item $ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}$
  \item $T(n) = (2\lceil n/k \rceil - 1)(M(k) + 2k - 1) + T(n - k)$
\end{itemize}
\[ \text{Analysis} \]

- \( ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3} \)
- \( T(n) = (2\lceil n/k \rceil - 1)(M(k) + 2k - 1) + T(n - k) \)

\[ T(n) \leq (2c + 7)M(n) + o(M(n)) \]
In-place short product

\[ k \leq \frac{n}{c+2} \times T(n) = \left\lceil \frac{n}{k} \right\rceil M(k) + (\left\lceil \frac{n}{k} \right\rceil - 1) M(k-1) + 2k (\left\lceil \frac{n}{k} \right\rceil - 1) + T(n-k) \leq (2c+5) M(n) + o(M(n)) \]
In-place short product

\[ k \times ck = \frac{n}{c+2} \cdot T(n) = \left\lceil \frac{n}{k} \right\rceil M(k) + (\left\lceil \frac{n}{k} \right\rceil - 1) M(k-1) + 2k (\left\lceil \frac{n}{k} \right\rceil - 1) + T(n-k) \leq (2c+5) M(n) + o(M(n)) \]
In-place short product

\[ k \times \frac{n}{k} \leq \frac{n}{c+2} \cdot T(n) = \left\lceil \frac{n}{k} \right\rceil M(k) + \left( \left\lceil \frac{n}{k} \right\rceil - 1 \right) M(k-1) + 2k \left( \left\lceil \frac{n}{k} \right\rceil - 1 \right) + T(n-k) \leq (2c+5)M(n) + o(M(n)) \]
In-place short product

\[ k \leq \frac{n}{c+2} \]

\[ T(n) = \left\lceil \frac{n}{k} \right\rceil M(k) + \left( \left\lceil \frac{n}{k} \right\rceil - 1 \right) M(k-1) + 2k \left( \left\lceil \frac{n}{k} \right\rceil - 1 \right) + T(n-k) \]

\[ T(n) \leq \left( 2c+5 \right) M(n) + o(M(n)) \]
In-place short product

\[ k \times \left\lfloor \frac{n}{k} \right\rfloor \leq \frac{n}{(c+2)} \quad \text{T}(n) = \left\lfloor \frac{n}{k} \right\rfloor M(k) + (\left\lfloor \frac{n}{k} \right\rfloor - 1) M(k-1) + 2k \left( \left\lfloor \frac{n}{k} \right\rfloor - 1 \right) + T(n-k) \leq (2c+5) M(n) + o(M(n)) \]
In-place short product

- $k \leq n/(c + 2)$
- $T(n) = \lceil n/k \rceil M(k) + (\lceil n/k \rceil - 1)M(k-1) + 2k(\lceil n/k \rceil - 1) + T(n-k)$
In-place short product

- $k \leq \frac{n}{c + 2}$
- $T(n) = \left\lceil \frac{n}{k} \right\rceil M(k) + (\left\lceil \frac{n}{k} \right\rceil - 1)M(k-1) + 2k(\left\lceil \frac{n}{k} \right\rceil - 1) + T(n-k)$

$$T(n) \leq (2c + 5)M(n) + o(M(n))$$
In-place middle product

\[ \begin{array}{c}
\times \\
= \\
\end{array} \]
In-place middle product

\[ k \times \left\lceil \frac{n}{k} \right\rceil = \]

- Recursive call on part of \( f \)...
- But on full \( g \)!

\[ T(n, m) = \left\lceil \frac{n}{k} \right\rceil M(k) + T(n, m-k) \leq \begin{cases} M(n) \log c + 2c + 1(n) + o(M(n) \log n) & \text{if } M(n) \text{ is quasi-linear} \\ O(M(n)) & \text{otherwise} \end{cases} \]
In-place middle product

\[
\begin{align*}
\text{In-place middle product} & = k \left\lceil \frac{n}{k} \right\rceil \\
& \times (n-k) \\
& \text{Recursive call on part of } f \ldots \text{ but on full } g!
\end{align*}
\]

\[
T(n, m) = \left\lceil \frac{n}{k} \right\rceil M(k) + T(n, m-k)
\]

\[
T(n, n) \leq \begin{cases} 
M(n) \log c + 2c + 1(n) + o(M(n) \log n) & \text{if } M(n) \text{ is quasi-linear} \\
O(M(n)) & \text{otherwise}
\end{cases}
\]
In-place middle product

- Recursive call on part of \( f \) . . . but on full \( g \) !
- \( T(n, m) = \lceil n/k \rceil M(k) + T(n, m - k) \)
In-place middle product

- Recursive call on part of $f \ldots$ but on full $g$!
- $T(n, m) = \lceil n/k \rceil M(k) + T(n, m - k)$

$$T(n, n) \leq \begin{cases} 
M(n) \log_{\frac{c+2}{c+1}}(n) + o(M(n) \log n) & \text{if } M(n) \text{ is quasi-linear} \\
O(M(n)) & \text{otherwise}
\end{cases}$$
Other operations

Work in progress!
Other operations

Work in progress!

- Use our in-place algorithms as building blocks
  - Newton iteration: division, square root, …
  - Evaluation & interpolation
    \[ \rightarrow (\text{at most}) \log(n) \text{ increase in complexity} \]
Other operations

Work in progress!

- Use our in-place algorithms as building blocks
  - Newton iteration: division, square root, ...  
  - Evaluation & interpolation

→ (at most) $\log(n)$ increase in complexity

Remark

- In place: division with remainder
- Only quotient or only remainder: not clear
- Main difficulty: size of the output
Summary

\[ O(1) \text{ or } \log n \]

\[ 2c + 7 \]

\[ 2c + 5 \]

\[ O(1) \text{ or } \log n \]
Conclusion

- TISP-reductions between polynomial products
- Self-reductions to obtain in-place algorithms
Conclusion

- TISP-reductions between polynomial products
- Self-reductions to obtain in-place algorithms

Comparisons

- Better use specialized in-place algorithms...
- ... when they exist!
Conclusion

- TISP-reductions between polynomial products
- Self-reductions to obtain in-place algorithms

Comparisons
- Better use specialized in-place algorithms...
- ... when they exist!

Main open problems
- Remove the $\log(n)$ for middle product or prove a lower bound
- General result on Tellegen’s transposition principle
- What about integer multiplication?
Conclusion

- TISP-reductions between polynomial products
- Self-reductions to obtain in-place algorithms

Comparisons
- Better use specialized in-place algorithms...
- ... when they exist!

Main open problems
- Remove the log$(n)$ for middle product or prove a lower bound
- General result on Tellegen’s transposition principle
- What about integer multiplication?

Thank you!