Memory-efficient polynomial arithmetic

Pascal Giorgi\textsuperscript{1}  Bruno Grenet\textsuperscript{1}  Daniel S. Roche\textsuperscript{2}

\textsuperscript{1} LIRMM, Université de Montpellier

\textsuperscript{2} CS Department, US Naval Academy
Multiplication of polynomials

- **Input.** \( F = \sum_{i=0}^{n-1} F[i]X^i \) and \( G = \sum_{j=0}^{n-1} G[j]X^j \)
- **Output.** \( H = F \times G = \sum_{k=0}^{2n-2} H[k]X^k \)
Multiplication of polynomials

- **Input.** $F = \sum_{i=0}^{n-1} F[i]X^i$ and $G = \sum_{j=0}^{n-1} G[j]X^j$
- **Output.** $H = F \times G = \sum_{k=0}^{2n-2} H[k]X^k$

For $i = 0$ to $n-1$:
   For $j = 0$ to $n-1$:
      $H[i+j] += F[i]*G[j]$
**Multiplication of polynomials**

- **Input.** $F = \sum_{i=0}^{n-1} F[i]X^i$ and $G = \sum_{j=0}^{n-1} G[j]X^j$
- **Output.** $H = F \times G = \sum_{k=0}^{2n-2} H[k]X^k$

For $i = 0$ to $n-1$:
  
  For $j = 0$ to $n-1$:
  
  $H[i+j] += F[i] \times G[j]$

- **Karatsuba’s algorithm:**
  
  $\left( f_0 + X^{\frac{n}{2}} f_1 \right) \cdot \left( g_0 + X^{\frac{n}{2}} g_1 \right)$
  
  $= f_0 g_0 + \left( (f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1 \right) X^{\frac{n}{2}} + f_1 g_1 X^n$
Multiplication of polynomials

- **Input.** \( F = \sum_{i=0}^{n-1} F[i]X^i \) and \( G = \sum_{j=0}^{n-1} G[j]X^j \)
- **Output.** \( H = F \times G = \sum_{k=0}^{2n-2} H[k]X^k \)

For \( i = 0 \) to \( n-1 \):
  For \( j = 0 \) to \( n-1 \):
    \( H[i+j] += F[i]*G[j] \)

- **Karatsuba’s algorithm:** \( (f_0 + X^{\frac{n}{2}}f_1) \cdot (g_0 + X^{\frac{n}{2}}g_1) = f_0g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0g_0 - f_1g_1)X^{\frac{n}{2}} + f_1g_1X^n \)

- **Toom-Cook algorithm:** split \( F \) and \( G \) in three or more parts
Multiplication of polynomials

- **Input.** \( F = \sum_{i=0}^{n-1} F[i]X^i \) and \( G = \sum_{j=0}^{n-1} G[j]X^j \)
- **Output.** \( H = F \times G = \sum_{k=0}^{2n-2} H[k]X^k \)

For \( i = 0 \) to \( n-1 \):
  For \( j = 0 \) to \( n-1 \):
    \( H[i+j] += F[i]*G[j] \)

- **Karatsuba’s algorithm:** \( (f_0 + X^{n/2}f_1) \cdot (g_0 + X^{n/2}g_1) \)
  \( = f_0g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0g_0 - f_1g_1)X^{n/2} + f_1g_1X^n \)

- **Toom-Cook algorithm:** split \( F \) and \( G \) in three or more parts

- **FFT-based algorithms:**
  \( (F, G) \xrightarrow{\text{eval.}} (F(\omega^i), G(\omega^i)); \xrightarrow{\text{mult.}} FG(\omega^i); \xrightarrow{\text{interp.}} FG \)
Time complexity of polynomial arithmetic

- **Multiplication: $M(n)$**
  - Naïve: $O(n^2)$
  - Karatsuba: $O(n^{\log_2 3}) = O(n^{1.585})$ \footnote{Karatsuba (1962)}
  - Toom-3: $O(n^{\log_3 5}) = O(n^{1.465})$ \footnote{Toom (1963), Cook (1966)}
  - FFT-based:
    - $O(n \log n)$ with $2n$-th root of unity \footnote{Cooley, Tukey (1965)}
    - $O(n \log n \log \log n)$ \footnote{Schönhage, Strassen (1971)}
Time complexity of polynomial arithmetic

- Multiplication: $M(n)$
  - Naïve: $O(n^2)$
  - Karatsuba: $O(n^\log_3^3) = O(n^{1.585})$ \text{Karatsuba (1962)}
  - Toom-3: $O(n^\log_3^5) = O(n^{1.465})$ \text{Toom (1963), Cook (1966)}
  - FFT-based:
    - $O(n \log n)$ with $2n$-th root of unity \text{Cooley, Tukey (1965)}
    - $O(n \log n \log \log n)$ \text{Schönhage, Strassen (1971)}

- Other tasks:
  - Euclidean division: $O(M(n))$
  - GCD: $O(M(n) \log n)$
  - Evaluation & interpolation: $O(M(n) \log n)$
  - ...
Time complexity of polynomial arithmetic

- **Multiplication:** $M(n)$
  - Naïve: $O(n^2)$
  - Karatsuba: $O(n^\log_2 3) = O(n^{1.585})$ \text{ Karatsuba (1962)}
  - Toom-3: $O(n^{\log_3 5}) = O(n^{1.465})$ \text{ Toom (1963), Cook (1966)}
  - FFT-based:
    - $O(n \log n)$ with $2n$-th root of unity \text{ Cooley, Tukey (1965)}
    - $O(n \log n \log \log n)$ \text{ Schönhage, Strassen (1971)}

- **Other tasks:**
  - Euclidean division: $O(M(n))$
  - GCD: $O(M(n) \log n)$
  - Evaluation & interpolation: $O(M(n) \log n)$
  - ...

What about space complexity?
First thought: count extra memory apart from input/output

- Naive algorithm: \( O(1) \)
- Karatsuba, Toom-3, FFT: \( O(n) \)
- Other tasks: often \( O(n) \), sometime \( O(n \log n) \)
First thought: count extra memory apart from input/output
- Naive algorithm: $O(1)$
- Karatsuba, Toom-3, FFT: $O(n)$
- Other tasks: often $O(n)$, sometime $O(n \log n)$

However, need to precise the complexity model !!!
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
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
  - *Reasonable* from a programmer’s viewpoint
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
  - *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
  - *Reasonable* from a programmer’s viewpoint

- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end
Karatsuba’s algorithm:

\[
\left(f_0 + X^{\frac{n}{2}} f_1 \right) \cdot \left(g_0 + X^{\frac{n}{2}} g_1 \right) = f_0 g_0 + (f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1)X^{\frac{n}{2}} + f_1 g_1 X^n
\]

with some intuition space of \(2n\)
Karatsuba’s algorithm:

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

with some intuition space of 2n

- Thomé (2002) : space of \( n + O(\log n) \)
  \( \rightarrow \) careful use output + \( n \) temp. registers + \( O(\log n) \) stack
Karatsuba’s algorithm:

\[( f_0 + X^{\frac{n}{2}} f_1 ) \cdot ( g_0 + X^{\frac{n}{2}} g_1 ) = f_0 g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1) X^{\frac{n}{2}} + f_1 g_1 X^n \]

with some intuition space of 2n

- **Thomé (2002)**: space of \( n + O(\log n) \)
  → careful use output + \( n \) temp. registers + \( O(\log n) \) stack

- **Roche (2009)**: space of only \( O(\log n) \)
  → half-additive version (\( h \leftarrow h_\ell + fg \) where \( \deg(h_\ell) < n \))
FFT-based algorithms:

\[(F, G) \rightarrow (F(\omega^i), G(\omega^i))_i \rightarrow FG(\omega^i)_i \rightarrow FG\]
Previous results

FFT-based algorithms:

\[(F, G) \rightarrow (F(\omega^i), G(\omega^i)) \rightarrow FG(\omega^i) \rightarrow FG\]

space of \(2n\) : FFT is in-place (overwriting) but \# points \(\approx 2n\)
FFT-based algorithms:

\[(F, G) \rightarrow (F(\omega^i), G(\omega^i))_i \rightarrow FG(\omega^i)_i \rightarrow FG\]

space of \(2n\): FFT is in-place (overwriting) but \# points \(\approx 2n\)

- **Roche (2009):** space of \(O(1)\) when \(n = 2^k\) and \(\omega^{2n} = 1\)
  \(\rightarrow\) compute half of the result + recurse

- **Harvey-Roche (2010):** space of \(O(1)\) when \(\omega^{2n} = 1\)
### Previous results

#### Summary of complexities

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Time complexity</th>
<th>Space complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>naive</td>
<td>$2n^2 + 2n - 1$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Karatsuba ('62)</td>
<td>$&lt; 6.5n^\log(3)$</td>
<td>$\leq 2n + 5 \log(n)$</td>
</tr>
<tr>
<td>Karatsuba (Thomé’02)</td>
<td>$&lt; 7n^\log(3)$</td>
<td>$\leq n + 5 \log(n)$</td>
</tr>
<tr>
<td>Karatsuba (Roche’09)</td>
<td>$&lt; 10n^\log(3)$</td>
<td>$\leq 5 \log(n)$</td>
</tr>
<tr>
<td>Toom-3 ('63)</td>
<td>$&lt; \frac{73}{4} n^\log_3(5)$</td>
<td>$\leq 2n + 5 \log_3(n)$</td>
</tr>
<tr>
<td>FFT (CT’65)</td>
<td>$9n \log(2n) + O(n)$</td>
<td>$2n$</td>
</tr>
<tr>
<td>FFT (Roche’09)</td>
<td>$11n \log(2n) + O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>TFT (HR’10)</td>
<td>$O(n \log(n))$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
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)
Polynomial products and linear maps

Space-preserving reductions

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

\[ \text{Short product} \]

- Product of truncated power series
- Useful in other algorithms
- Time complexity: \( M(n) \)
- Space complexity: \( O(n) \)
Short product

\[ n \times n - 1 = \text{low short product} \times \text{high short product} \]

- Low short product: product of truncated power series
- Useful in other algorithms
- Time complexity: \( M(n) \)
- 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

\[ \begin{array}{c}
\times \\
\end{array} \]

Useful for Newton iteration

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

Division, square root, ...

Time complexity: \( M(n) \)

Space complexity: \( O(n) \)

12
Middle product

\[ \text{middle product} = n - 1 \times 2n - 1 \]

- Useful for Newton iteration
- \( G \leftarrow G \left( 1 - GF \right) \text{ mod } X \)
- division, square root, ...
- Time complexity: \( M(n) \)
- Space complexity: \( O(n) \)

Tellegen's transposition
Middle product

- Useful for Newton iteration
  - $G \leftarrow G(1 - GF) \mod X^{2n}$ with $GF = 1 + X^nH$
  - division, square root, ...

- Time complexity: $M(n) \rightarrow$ Tellegen’s transposition
- Space complexity: $O(n)$
Multiplications as linear maps

Example:

\[ f = 3X^2 + 2X + 1 \]
\[ g = X^2 + 2X + 4 \]
\[ fg = 3X^4 + 8X^3 + 17X^2 + 10X + 4 \]
Multiplications as linear maps

Example:

\[ f = 3X^2 + 2X + 1 \]
\[ g = X^2 + 2X + 4 \]
\[ fg = 3X^4 + 8X^3 + 17X^2 + 10X + 4 \]

\[
\begin{bmatrix}
1 & 0 & 0 \\
2 & 1 & 0 \\
3 & 2 & 1 \\
3 & 2 & 3
\end{bmatrix}
\begin{bmatrix}
1 \\
4 \\
2 \\
1
\end{bmatrix}
= 
\begin{bmatrix}
4 \\
10 \\
17 \\
8 \\
3
\end{bmatrix}
\]
Multiplications as linear maps

Full product:

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

Short products:

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

Middle product:

\[ \times = 3n - 1 \]
Middle product:
For simplicity in the presentation we assume

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

- Without space restrictions:
  - $\text{SP} \leq \text{FP}$ and $\text{FP} \leq \text{SP}_{lo} + \text{SP}_{hi}$
  - $\text{MP} \equiv \text{FP}$ (transposition)
  - $\text{MP} \leq \text{SP}_{lo} + \text{SP}_{hi} + (n - 1)$ 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:
  - \( FP : (n, n) \rightarrow 2n - 1 \)
  - \( SP_{lo} : (n, n) \rightarrow n \)
  - \( SP_{hi} : (n - 1, n - 1) \rightarrow n - 1 \)
  - \( MP : (2n - 1, n) \rightarrow n \)
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:
  - \( FP : (n, n) \rightarrow 2n - 1 \)
  - \( SP_{lo} : (n, n) \rightarrow n \)
  - \( SP_{hi} : (n - 1, n - 1) \rightarrow n - 1 \)
  - \( MP : (2n - 1, n) \rightarrow n \)

\( \times \) Reductions unusable in space-restricted settings!
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:
  - $FP : (n, n) \rightarrow 2n - 1$
  - $SP_{lo} : (n, n) \rightarrow n$
  - $SP_{hi} : (n - 1, n - 1) \rightarrow n - 1$
  - $MP : (2n - 1, n) \rightarrow n$

✗ Reductions unusable in space-restricted settings!

✓ We provide space/time preserving reductions
A relevant notion of reduction

Definitions

- $\text{TISP}(t(n), s(n))$: computable in time $t(n)$ and space $s(n)$
- $A \leq_c B$: $A$ is computable with oracle $B$
  
  if $B \in \text{TISP}(t(n), s(n))$ then

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

- $A \equiv_c B$: $A \leq_c B$ and $B \leq_c A$
A relevant notion of reduction

Definitions

- TISP\((t(n), s(n))\): computable in time \(t(n)\) and space \(s(n)\)
- \(A \leq_c B\): \(A\) is computable with oracle \(B\)
  
  if \(B \in \text{TISP}(t(n), s(n))\) then

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

- \(A \equiv_c B\): \(A \leq_c B\) and \(B \leq_c A\)

Example

\(A \equiv_1 B\) means \(A\) and \(B\) are equivalent for both time and space
First results in a nutshell

Theorem

FP \leq 2 \leq 1 \equiv 1

SP_{lo} \parallel_{1} \leq 1

SP_{hi}

MP
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!)

- $SP_{lo}(n) \leq MP(n)$; $SP_{hi}(n) \leq MP(n - 1)$

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

Remark $FP^+_{lo} \equiv_1 FP^+_{hi}$ using reversal polynomials
Half-additive full product: $h \leftarrow h + f \cdot g$

Remark $\text{FP}_\text{lo}^+ \equiv_1 \text{FP}_\text{hi}^+$ using reversal polynomials

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

$ FP \times \lo(n) \leq SP\lo(n) + SP\hi(n) + n - 1 $
From SP to FP$^+$

$$\text{FP}_{lo}^+(n) \leq \text{SP}_{lo}(n) + \text{SP}_{hi}(n) + n - 1$$
\[(f_0 + X^{[n/2]} f_1) \cdot (g_0 + X^{[n/2]} g_1) = f_0 g_0 + X^{[n/2]} (f_0 g_1 + f_1 g_0) \pmod{X^n}\]
\[(f_0 + X^{\lceil n/2 \rceil} f_1) \cdot (g_0 + X^{\lceil n/2 \rceil} g_1) = f_0 g_0 + X^{\lceil n/2 \rceil} (f_0 g_1 + f_1 g_0) \mod X^n\]
From FP$^+$ to SP

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

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

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

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

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

- From SP to MP:
  - partial result:
    - up to \( \log(n) \) increase in time complexity
    - techniques from next part
  - without space restriction
    - FP to MP through Tellegen’s transposition principle
Summary of results so far

\[
\begin{align*}
&\text{iSP} \\
&\text{iMP} & 1 \\
&\text{iFP} & 1 \\
&\text{iFP}^+ \\
&\text{oSP} \\
&\text{oMP} & \leq 1 \\
&\text{oFP} & \frac{m}{n} \\
&\text{oFP}_u^+ 
\end{align*}
\]
In-place algorithms from out-of-place algorithms
Framework

- In-place algorithms parametrized by out-of-place algorithm
  - Out-of-place: uses $cn$ extra space
  - Constant $c$ known to the algorithm

Similar approach for matrix mul.: Boyer, Dumas, Pernet, Zhou (2009)
- **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
  - *Fake* padding
  - *Tail* recursive call

Similar approach for matrix mul.:
   - Boyer, Dumas, Pernet, Zhou (2009)
Framework

- 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
  - Fake padding
  - Tail recursive call

Similar approach for matrix mul.: Boyer, Dumas, Pernet, Zhou (2009)
Tail recursion and fake padding

- Tail recursion:
  - Only one recursive call + last (or first) instruction
  - No need of recursive stack $\iff$ 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 \(\Rightarrow\) 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!
Our results

- In-place full product (half additive) in time $(2c + 7)M(n)$
- In-place short product in time $(2c + 5)M(n)$
- In-place middle product in time $O(M(n) \log n)$
In-place FP$^+$ from out-of-place FP

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

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

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$
\[(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}\]
\[
\left\lceil \frac{n}{k} \right\rceil - 1 \times (n - k) = \left\lceil \frac{n}{k} \right\rceil \cdot (M(k) + 2k - 1) + T(n - k) \leq 2(c + 7)M(n) + o(M(n))
\]
- $ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}$
- $T(n) = (2\left\lceil \frac{n}{k} \right\rceil - 1)(M(k) + 2k - 1) + T(n - k)$
\[
ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}
\]

\[
T(n) = (2\lceil n/k \rceil - 1)(M(k) + 2k - 1) + T(n - k)
\]

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

\[
\begin{align*}
\text{\(k \leq n/(c+2)\)} \\
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 (2c+5) M(n) + o(M(n))
\end{align*}
\]
In-place short product

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

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

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

\[ \begin{align*}
\text{In-place short product} & = k \times \left\lceil \frac{n}{k} \right\rceil \\
& \leq n / (c + 2) \\
T(n) & = \left\lceil \frac{n}{k} \right\rceil M(k) + \left( \left\lceil \frac{n}{k} \right\rceil - 1 \right) M(k - 1) + 2k \left( \left\lceil \frac{n}{k} \right\rceil - 1 \right) + T(n - k)
\end{align*} \]
In-place short product

\[ \left\lceil \frac{n}{k} \right\rceil \times k \leq \frac{n}{c+2} \]

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

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

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

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

\[ \lceil \frac{n}{k} \rceil \times k = T(n, m) \]

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

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

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

Recursive call on chunks of size \(f\) but with full size \(g\)!

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

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

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

- Recursive call on chunks of \( f \) . . . but with 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
→ (at most) \( \log(n) \) increase in complexity
Other operations

Work in progress!

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

Remark

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

\[ \log (2c + 5) \leq \frac{3}{2} \]

\[ \log (2c + 7) \leq \frac{m}{n} \leq 1 \]

\[ \leq 2c + 7 \]
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!