Memory-efficient polynomial arithmetic

Pascal Giorgi[1] Bruno Grenet[1] Daniel S. Roche[2]

[1] LIRMM, Université de Montpellier

[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]*G[j]
```

- Karatsuba's algorithm: $\left(f_0 + X^{\frac{n}{2}} f_1\right) \cdot \left(g_0 + X^{\frac{n}{2}} g_1\right)$

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

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

- 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: $\left(f_0 + X^{\frac{n}{2}} f_1\right) \cdot \left(g_0 + X^{\frac{n}{2}} g_1\right)$
  $= f_0 g_0 + ((f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1)X^{\frac{n}{2}} + f_1 g_1 X^n$
- Toom-Cook algorithm: split F and G in three or more parts
- FFT-based algorithms:
  $(F, G) \xrightarrow{\text{eval.}} (F(\omega^i), G(\omega^i))_i \xrightarrow{\text{mult.}} FG(\omega^i)_i \xrightarrow{\text{interp.}} FG$

Time complexity of polynomial arithmetic

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

Time complexity of polynomial arithmetic

- Multiplication: $\mathsf{M}(n)$
    - Naïve: $O(n^2)$
    - Karatsuba: $O(n^{\log_2 3}) = O(n^{1.585})$ Karatsuba (1962)
    - Toom-3: $O(n^{\log_3 5}) = O(n^{1.465})$ Toom (1963), Cook (1966)
    - FFT-based:
        - $O(n \log n)$ with $2n$-th root of unity Cooley, Tukey (1965)
        - $O(n \log n \log\log n)$ Schönhage, Strassen (1971)
- Other tasks:
    - Euclidean division: $O(\mathsf{M}(n))$
    - GCD: $O(\mathsf{M}(n) \log n)$
    - Evaluation & interpolation: $O(\mathsf{M}(n) \log n)$
    - . . .

Time complexity of polynomial arithmetic

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

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

What about space complexity?

Space complexity of polynomial arithmetic

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 $O(n \log n)$

Space complexity of polynomial arithmetic

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:

- $\rightarrow$ *Standard* registers of size $O(\log n)$
- $\rightarrow$ *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 × space for multiplication

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 × 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 × 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 × 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

Previous results

Karatsuba's algorithm:

$$\left(f_0+X^{\frac{n}{2}}f_1\right)\cdot\left(g_0+X^{\frac{n}{2}}g_1\right)=f_0g_0+((f_0+f_1)(g_0+g_1)-f_0g_0-f_1g_1)X^{\frac{n}{2}}+f_1g_1X^n$$

with some intuition space of $2n$

Previous results

Karatsuba's algorithm:

$$\left(f_0+X^{\frac{n}{2}}f_1\right)\cdot\left(g_0+X^{\frac{n}{2}}g_1\right)=f_0g_0+((f_0+f_1)(g_0+g_1)-f_0g_0-f_1g_1)X^{\frac{n}{2}}+f_1g_1X^n$$

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

Previous results

Karatsuba's algorithm:

$$\left(f_0+X^{\frac{n}{2}}f_1\right)\cdot\left(g_0+X^{\frac{n}{2}}g_1\right)=f_0g_0+((f_0+f_1)(g_0+g_1)-f_0g_0-f_1g_1)X^{\frac{n}{2}}+f_1g_1X^n$$

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

Previous results

FFT-based algorithms:

$$(F, G) \to (F(\omega^i), G(\omega^i))_i \to FG(\omega^i)_i \to FG$$

Previous results

FFT-based algorithms:

$$(F, G) \to (F(\omega^i), G(\omega^i))_i \to FG(\omega^i)_i \to FG$$

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

Previous results

FFT-based algorithms:

$$(F, G) \to (F(\omega^i), G(\omega^i))_i \to FG(\omega^i)_i \to FG$$

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

- Roche (2009): space of $O(1)$ when $n = 2^k$ and $\omega^{2n} = 1$
  → compute half of the result + recurse
- Harvey-Roche (2010): space of $O(1)$ when $\omega^{2n} = 1$
  → same with TFT v.d. Hoeven (2004)

Previous results

Summary of complexities

Algorithms	**Time complexity**	**Space complexity**
naive	$2n^2 + 2n - 1$	$O(1)$
Karatsuba ('62)	$< 6.5n^{\log(3)}$	$\leq 2n + 5\log(n)$
Karatsuba (Thomé'02)	$< 7n^{\log(3)}$	$\leq n + 5\log(n)$
Karatsuba (Roche'09)	$< 10n^{\log(3)}$	$\leq 5\log(n)$
Toom-3 ('63)	$< \frac{73}{4}n^{\log_3(5)}$	$\leq 2n + 5\log_3(n)$
FFT (CT'65)	$9n\log(2n) + O(n)$	$2n$
FFT (Roche'09)	$11n\log(2n) + O(n)$	$O(1)$
TFT (HR'10)	$O(n\log(n))$	$O(1)$

Our problematic

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

Our problematic

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

- $O(1)$-space Karatsuba's algorithm?
- What about Toom-Cook algorithm?

Our problematic

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)?

Our problematic

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

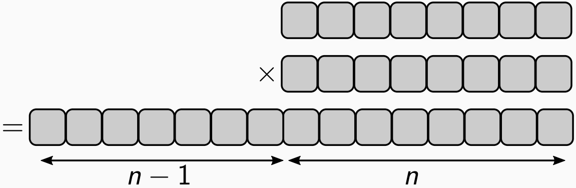
Polynomial products and linear maps

Space-preserving reductions

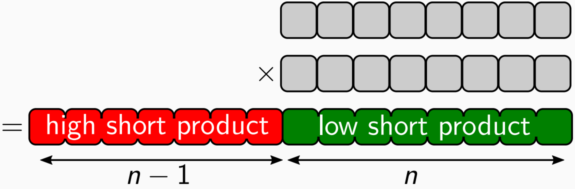
In-place algorithms from out-of-place algorithms

Polynomial products and linear maps

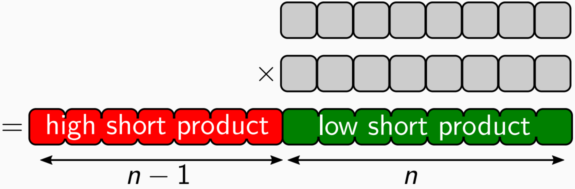
Short product



Short product

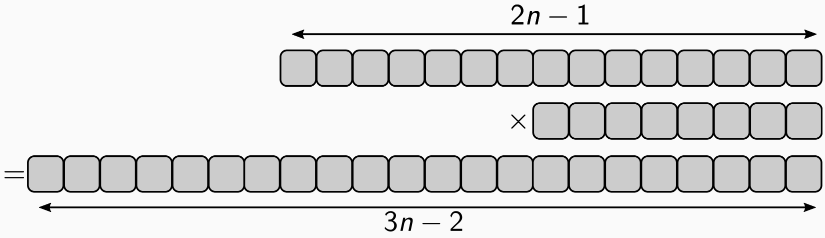


Short product

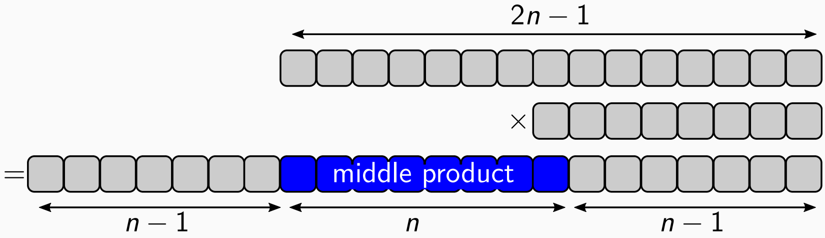


- Low short product: product of truncated power series
- Useful in other algorithms
- Time complexity: $\mathrm{M}(n)$
- Space complexity: $O(n)$

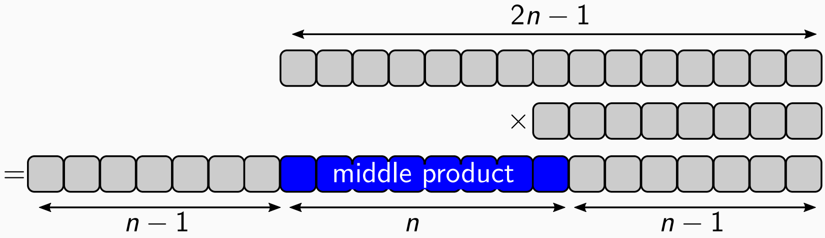
Middle product



Middle product



Middle product



- Useful for Newton iteration
    - $G \leftarrow G(1-GF) \bmod X^{2n}$ with $GF = 1 + X^n H$
    - division, square root, ...
- Time complexity: $\mathsf{M}(n) \rightarrow$ Tellegen's transposition
- Space complexity: $O(n)$

Multiplications as linear maps

Example:

$$
\begin{aligned}
f &= 3X^2 + 2X + 1 \\
g &= X^2 + 2X + 4 \\
fg &= 3X^4 + 8X^3 + 17X^2 + 10X + 4
\end{aligned}
$$

Multiplications as linear maps

Example:

$$f = 3X^2 + 2X + 1$$

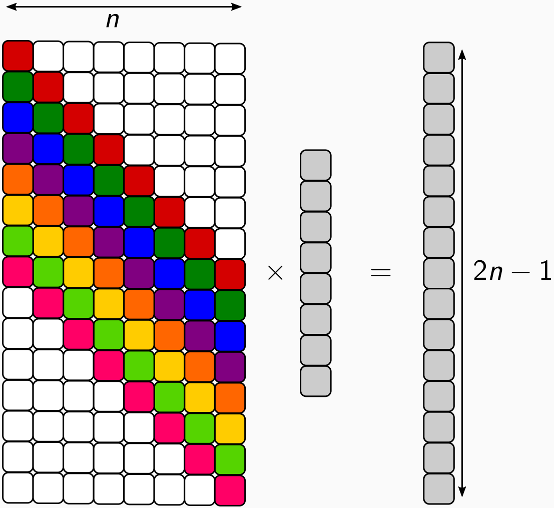
$$g = X^2 + 2X + 4$$

$$fg = 3X^4 + 8X^3 + 17X^2 + 10X + 4$$

$$\begin{bmatrix} 1 & & \\ 2 & 1 & \\ 3 & 2 & 1 \\ & 3 & 2 \\ & & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 10 \\ 17 \\ 8 \\ 3 \end{bmatrix}$$

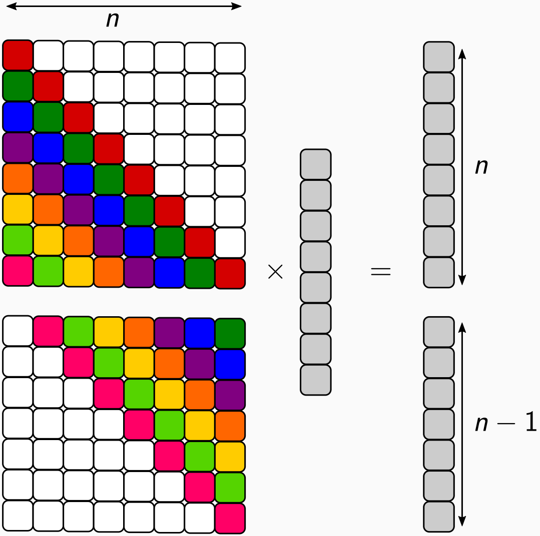
Multiplications as linear maps

Full product:



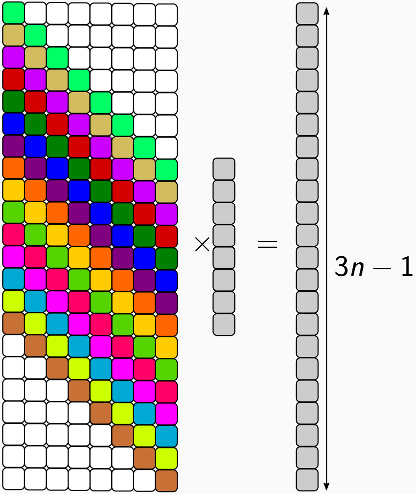
Multiplications as linear maps

Short products:



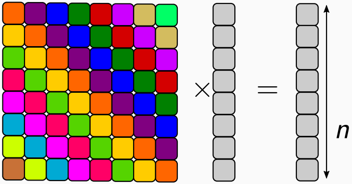
Multiplications as linear maps

Middle product:



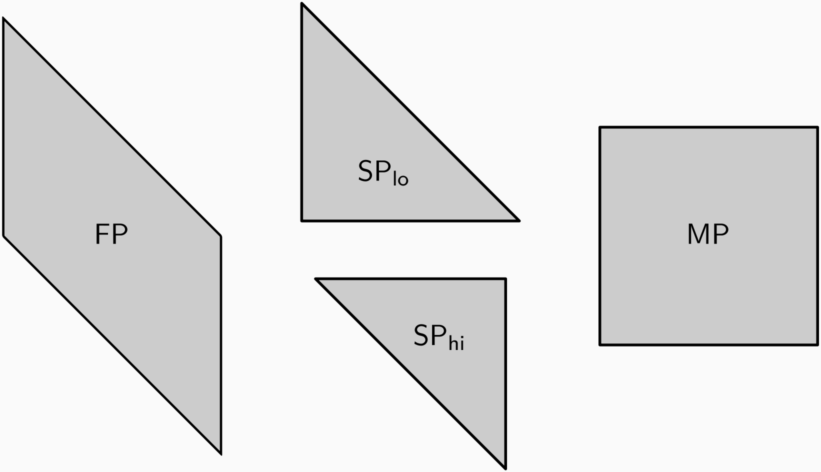
Multiplications as linear maps

Middle product:



Multiplications as linear maps

For simplicity in the presentation we assume



Full product Short products Middle product

Space-preserving reductions

Relative difficulties of products

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

Relative difficulties of products

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

- Size of inputs and outputs:
    - FP : $(n, n) \rightarrow 2n-1$
    - $\mathrm{SP}_{\mathrm{lo}}$: $(n, n) \rightarrow n$
    - $\mathrm{SP}_{\mathrm{hi}}$: $(n-1, n-1) \rightarrow n-1$
    - MP : $(2n-1, n) \rightarrow n$

Relative difficulties of products

- Without space restrictions:
  - $\mathsf{SP} \leq \mathsf{FP}$ and $\mathsf{FP} \leq \mathsf{SP}_{\mathsf{lo}} + \mathsf{SP}_{\mathsf{hi}}$
  - $\mathsf{MP} \equiv \mathsf{FP}$ (transposition)
  - $\mathsf{MP} \leq \mathsf{SP}_{\mathsf{lo}} + \mathsf{SP}_{\mathsf{hi}} + (n-1)$ additions
- Size of inputs and outputs:
  - $\mathsf{FP} : (n, n) \rightarrow 2n-1$
  - $\mathsf{SP}_{\mathsf{lo}} : (n, n) \rightarrow n$
  - $\mathsf{SP}_{\mathsf{hi}} : (n-1, n-1) \rightarrow n-1$
  - $\mathsf{MP} : (2n-1, n) \rightarrow n$

✗ Reductions unusable in space-restricted settings!

Relative difficulties of products

- Without space restrictions:
    - $\mathsf{SP} \leq \mathsf{FP}$ and $\mathsf{FP} \leq \mathsf{SP}_{\mathsf{lo}} + \mathsf{SP}_{\mathsf{hi}}$
    - $\mathsf{MP} \equiv \mathsf{FP}$ (transposition)
    - $\mathsf{MP} \leq \mathsf{SP}_{\mathsf{lo}} + \mathsf{SP}_{\mathsf{hi}} + (n-1)$ additions
- Size of inputs and outputs:
    - $\mathsf{FP} : (n, n) \to 2n-1$
    - $\mathsf{SP}_{\mathsf{lo}} : (n, n) \to n$
    - $\mathsf{SP}_{\mathsf{hi}} : (n-1, n-1) \to n-1$
    - $\mathsf{MP} : (2n-1, n) \to n$

✗ Reductions unusable in space-restricted settings!

✓ We provide space/time preserving reductions

A relevant notion of reduction

Definitions

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

  if $B \in \mathrm{TISP}(t(n), s(n))$ then

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

- $A \equiv_c B$: $A \leq_c B$ and $B \leq_c A$

A relevant notion of reduction

Definitions

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

  if $B \in \mathrm{TISP}(t(n), s(n))$ then

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

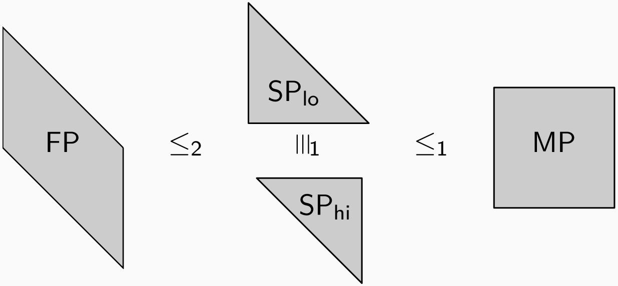
- $A \equiv_c B$: $A \leq_c B$ and $B \leq_c A$

Example

$A \equiv_1 B$ means A and B are equivalent for both time and space

First results in a nutshell

Theorem



$\mathrm{FP} \le_2 \mathrm{SP}_{\mathrm{lo}}$

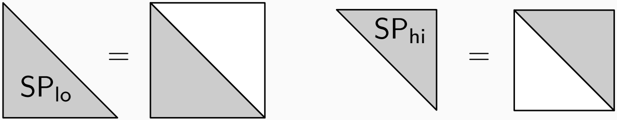
$|||_1$

$\mathrm{SP}_{\mathrm{hi}} \le_1 \mathrm{MP}$

Visual proof

Use of *fake padding* (in input, **not** in output!)

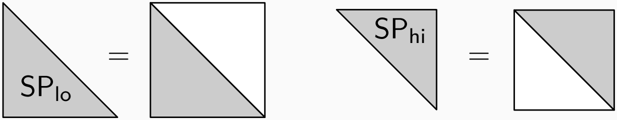
- $\mathrm{SP}_{\mathrm{lo}}(n) \leq \mathrm{MP}(n)$; $\mathrm{SP}_{\mathrm{hi}}(n) \leq \mathrm{MP}(n-1)$



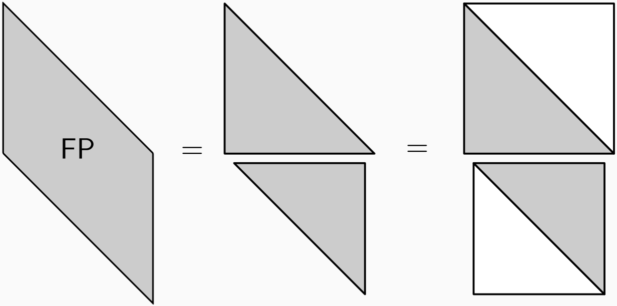
Visual proof

Use of *fake padding* (in input, **not** in output!)

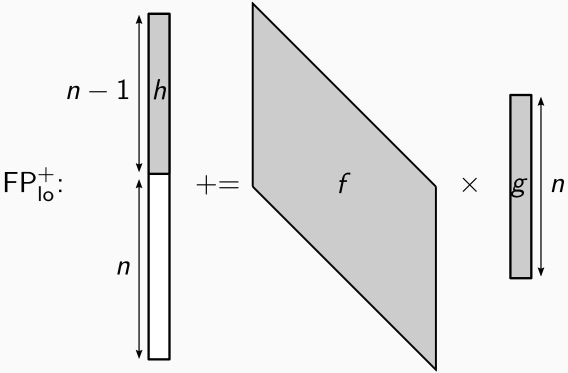
- $\mathrm{SP}_{\mathrm{lo}}(n) \leq \mathrm{MP}(n)$; $\mathrm{SP}_{\mathrm{hi}}(n) \leq \mathrm{MP}(n-1)$



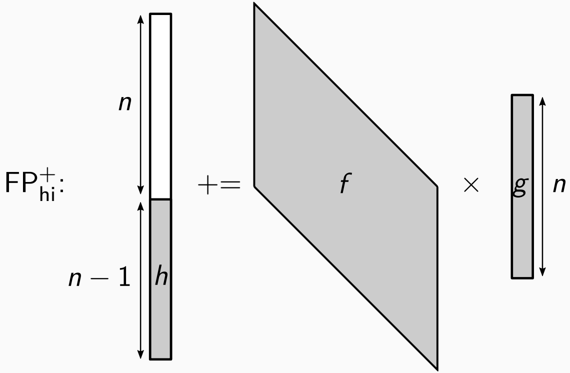
- $\mathrm{FP}(n) \leq \mathrm{SP}_{\mathrm{hi}}(n) + \mathrm{SP}_{\mathrm{lo}}(n) \leq \mathrm{MP}(n) + \mathrm{MP}(n-1)$



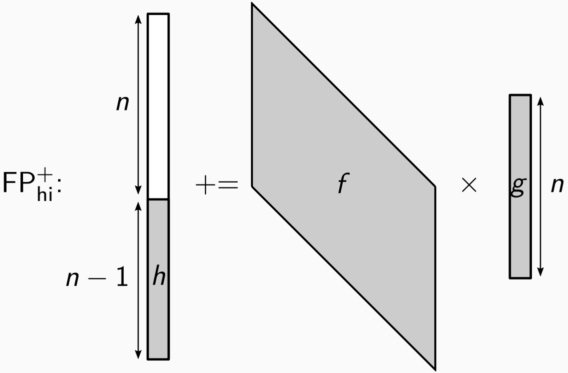
Half-additive full product: $h \leftarrow h + f \cdot g$



Half-additive full product: $h \leftarrow h + f \cdot g$

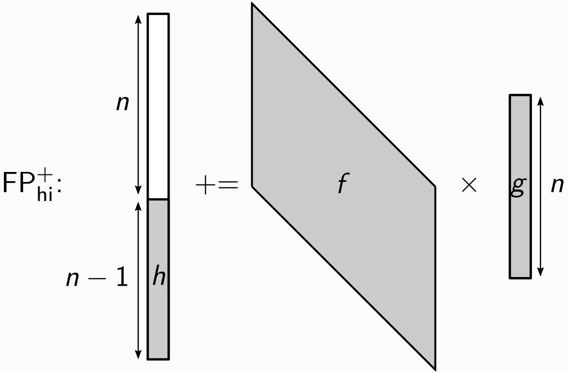


Half-additive full product: $h \leftarrow h + f \cdot g$



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

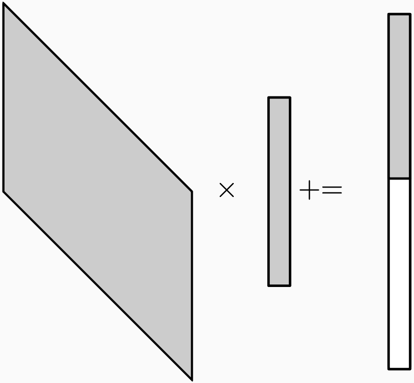
Half-additive full product: $h \leftarrow h + f \cdot g$



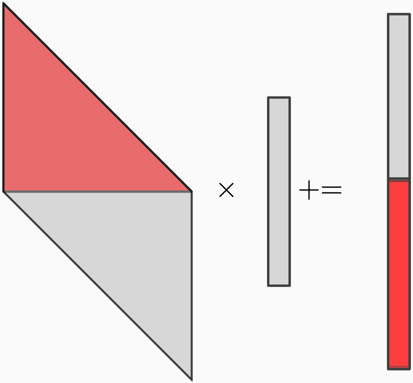
Remark $\mathsf{FP}^+_{\mathsf{lo}} \equiv_1 \mathsf{FP}^+_{\mathsf{hi}}$ using reversal polynomials

Theorem $\mathsf{FP}^+ \leq_2 \mathsf{SP}$ and $\mathsf{SP} \leq_{3/2} \mathsf{FP}^+$

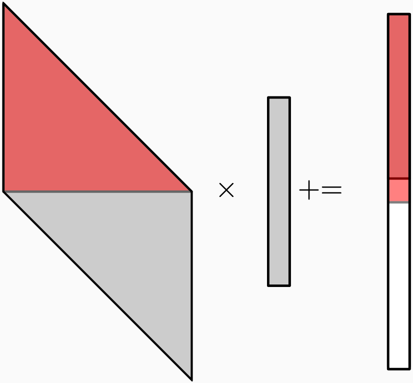
From SP to FP$^+$



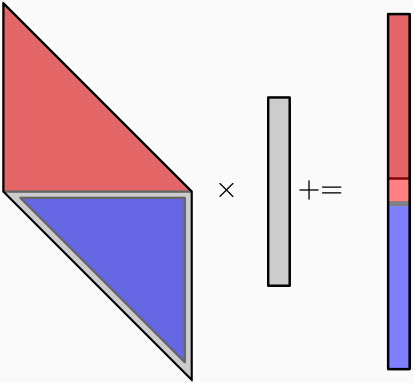
From SP to FP$^+$



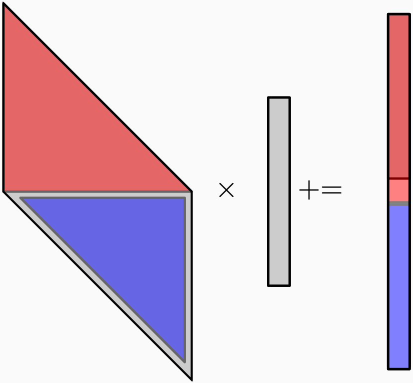
From SP to FP$^+$



From SP to FP$^+$



From SP to FP$^+$



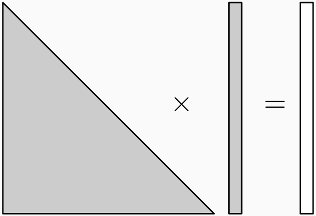
$$\mathsf{FP}^+_{\mathsf{lo}}(n) \leq \mathsf{SP}_{\mathsf{lo}}(n) + \mathsf{SP}_{\mathsf{hi}}(n) + n - 1$$

From FP$^+$ to SP

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

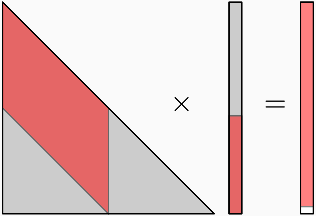
From FP^{+} to SP

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



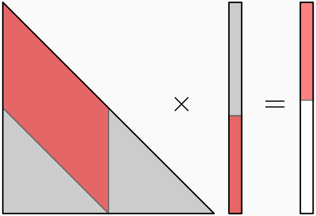
From FP$^+$ to SP

$$\left(f_0 + X^{\lceil n/2 \rceil} f_1\right) \cdot \left(g_0 + X^{\lceil n/2 \rceil} g_1\right) = f_0 g_0 + X^{\lceil n/2 \rceil}(\textcolor{red}{f_0 g_1} + f_1 g_0) \mod X^n$$



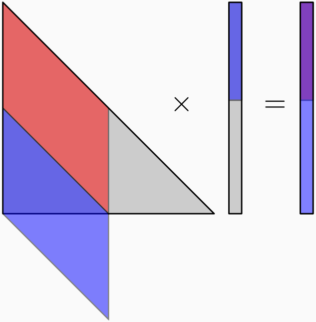
From FP$^+$ to SP

$$\left(f_0 + X^{\lceil n/2\rceil} f_1\right)\cdot\left(g_0 + X^{\lceil n/2\rceil} g_1\right) = f_0 g_0 + X^{\lceil n/2\rceil}(\color{red}{f_0 g_1} + f_1 g_0) \mod X^n$$



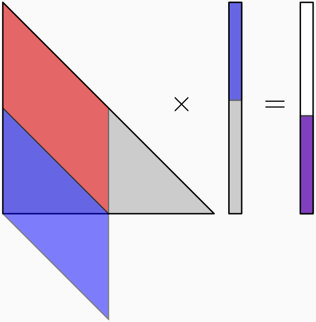
From FP$^+$ to SP

$$\left(f_0 + X^{\lceil n/2\rceil} f_1\right)\cdot\left(g_0 + X^{\lceil n/2\rceil} g_1\right) = f_0 g_0 + X^{\lceil n/2\rceil}\left(\color{red}{f_0 g_1} + \color{blue}{f_1 g_0}\right) \mod X^n$$



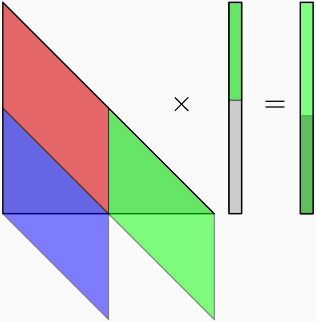
From FP$^+$ to SP

$$\left(f_0 + X^{\lceil n/2\rceil} f_1\right)\cdot\left(g_0 + X^{\lceil n/2\rceil} g_1\right) = f_0 g_0 + X^{\lceil n/2\rceil}\left(\textcolor{red}{f_0 g_1} + \textcolor{blue}{f_1 g_0}\right) \mod X^n$$



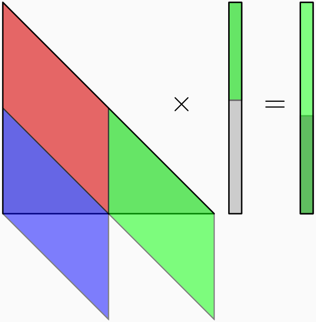
From FP$^+$ to SP

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



From FP$^+$ to SP

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



$$\mathrm{SP}_{\mathrm{lo}}(n) \leq \mathrm{FP}(\lfloor n/2 \rfloor) + \mathrm{FP}^+_{\mathrm{lo}}(\lfloor n/2 \rfloor) + \mathrm{FP}^+_{\mathrm{hi}}(\lceil n/2 \rceil)$$

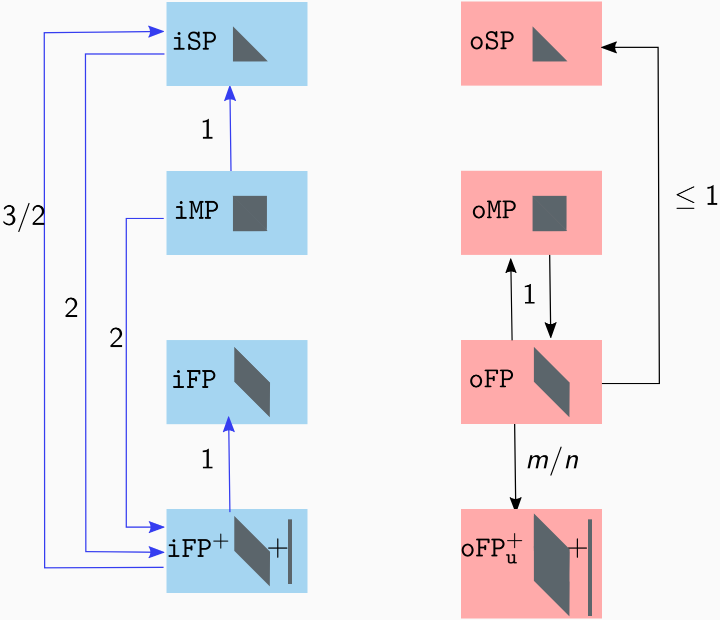
Converse directions?

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

Converse directions?

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



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

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

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

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 $\rightsquigarrow$ avoid $O(\log n)$ extra space

Tail recursion and fake padding

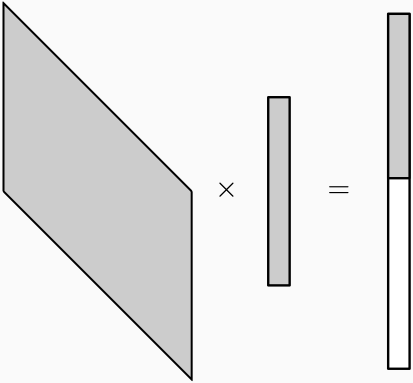
- Tail recursion:
    - Only one recursive call + last (or first) instruction
    - No need of recursive stack $\rightsquigarrow$ avoid $O(\log n)$ extra space
- *Fake* padding:
    - Pretend to pad inputs with zeroes
    - Make the data structure responsible for it
        - $O(1)$ increase in memory
        - *Cf.* strides in dense linear algebra
    - OK in inputs, not in outputs!

Our results

- In-place full product (half additive) in time $(2c+7)\mathsf{M}(n)$
- In-place short product in time $(2c+5)\mathsf{M}(n)$
- In-place middle product in time $O(\mathsf{M}(n)\log n)$

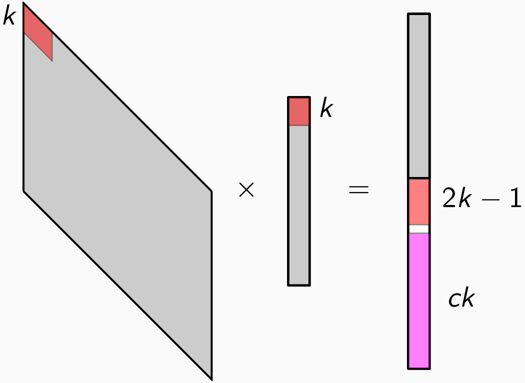
In-place FP^+ from out-of-place FP

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$



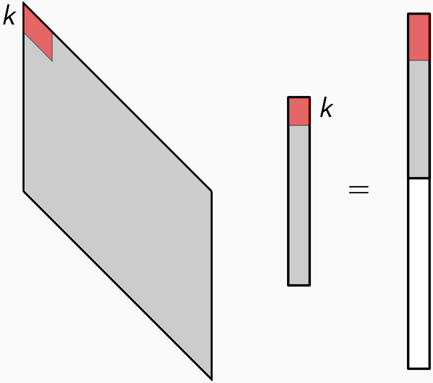
In-place FP$^+$ from out-of-place FP

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = \color{red}{f_0 g_0} \color{black} + X^k(f_0\hat{g} + \hat{f} g_0) + X^{2k}\hat{f}\hat{g}$$



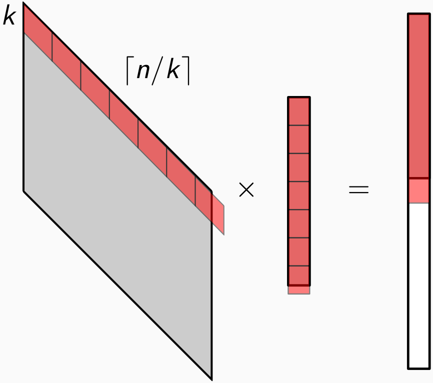
In-place FP$^+$ from out-of-place FP

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = {\color{red} f_0 g_0} + X^k(f_0\hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$



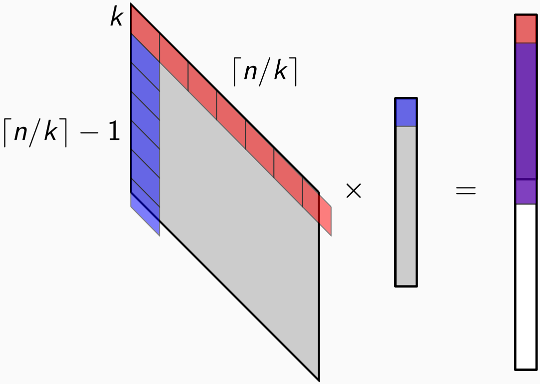
In-place FP$^+$ from out-of-place FP

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = \color{red}{f_0 g_0} + X^k (\color{red}{f_0 \hat{g}} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$



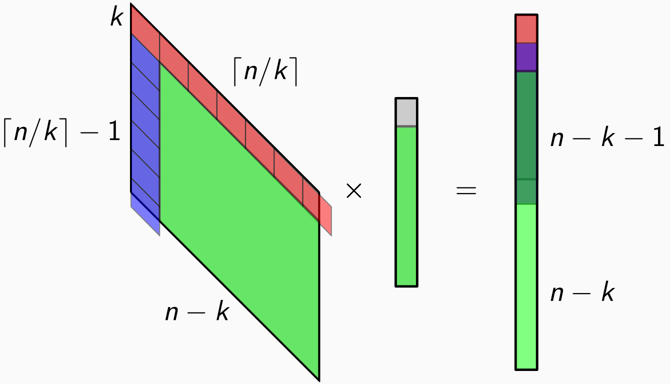
In-place FP$^+$ from out-of-place FP

$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = \color{red}{f_0 g_0} \color{black}{+ X^k(}\color{red}{f_0 \hat{g}} \color{black}{+} \color{blue}{\hat{f} g_0}\color{black}{) + X^{2k} \hat{f} \hat{g}}$$

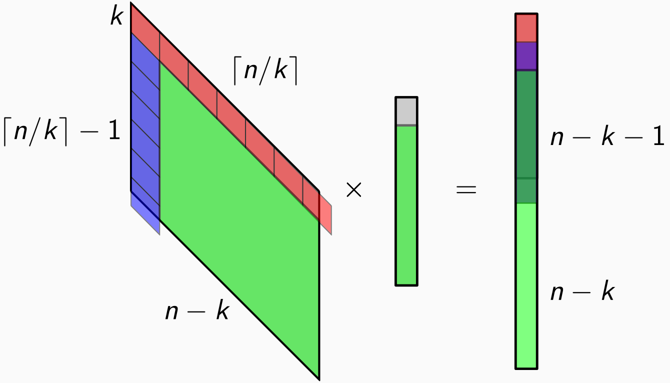


In-place FP$^+$ from out-of-place FP

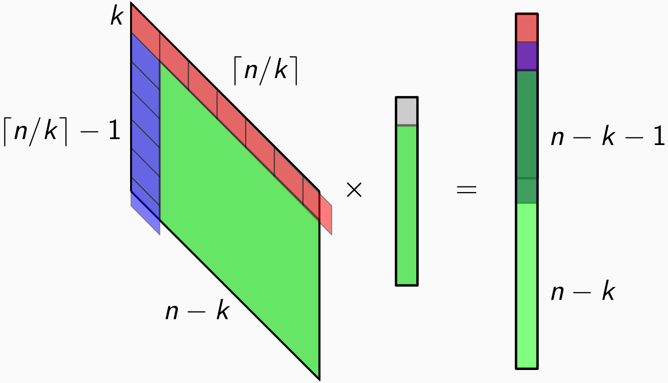
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = {\color{red} f_0 g_0} + X^k({\color{red} f_0 \hat{g}} + {\color{blue} \hat{f} g_0}) + X^{2k} {\color{green} \hat{f} \hat{g}}$$



Analysis

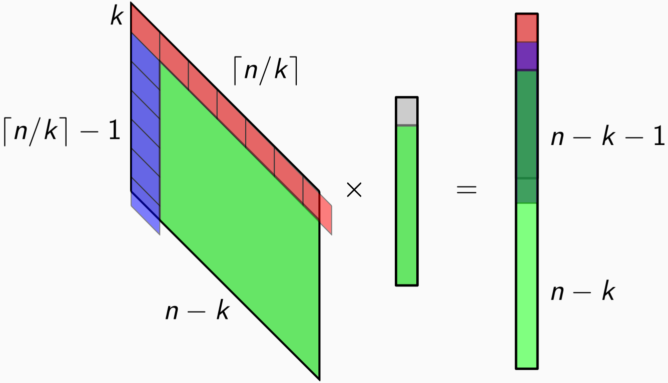


Analysis



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

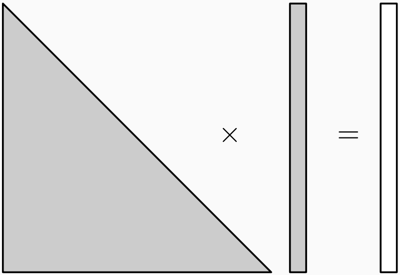
Analysis



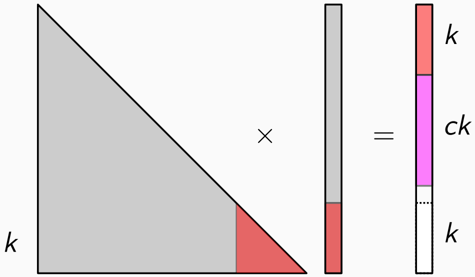
- $ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}$
- $T(n) = (2\lceil n/k \rceil - 1)(\mathrm{M}(k) + 2k - 1) + T(n-k)$

$$T(n) \leq (2c+7)\mathrm{M}(n) + o(\mathrm{M}(n))$$

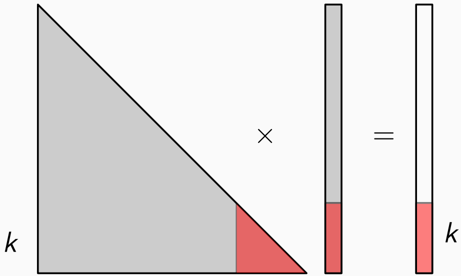
In-place short product



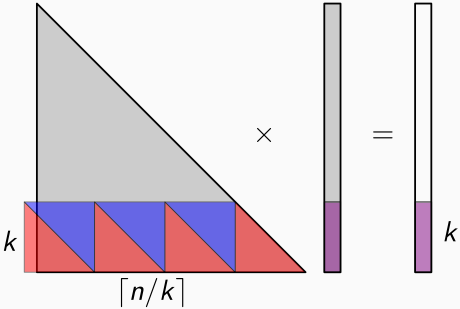
In-place short product



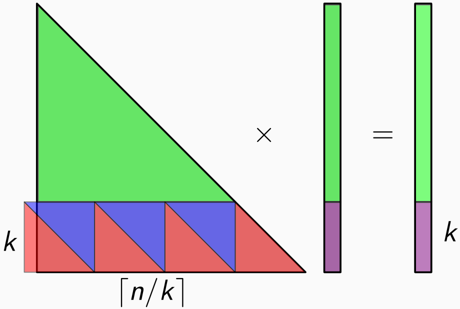
In-place short product



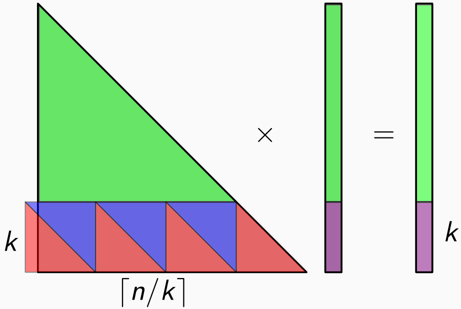
In-place short product



In-place short product

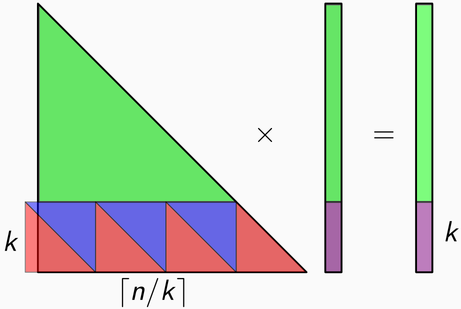


In-place short product



- $k \leq n/(c+2)$
- $T(n) = \lceil {}^{n}\!/_{k} \rceil \mathrm{M}(k) + (\lceil {}^{n}\!/_{k} \rceil - 1)\mathrm{M}(k-1) + 2k(\lceil {}^{n}\!/_{k} \rceil - 1) + T(n-k)$

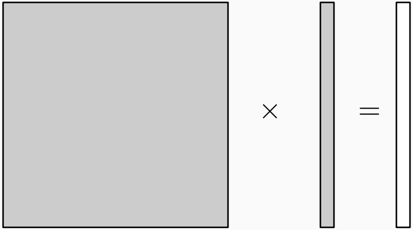
In-place short product



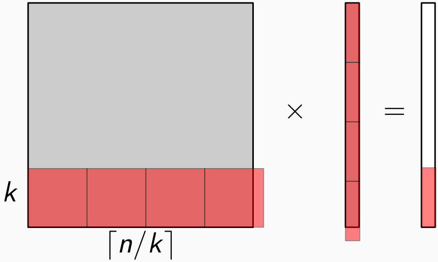
- $k \leq n/(c+2)$
- $T(n) = \lceil n/k \rceil \mathsf{M}(k) + (\lceil n/k \rceil - 1)\mathsf{M}(k-1) + 2k(\lceil n/k \rceil - 1) + T(n-k)$

$$T(n) \leq (2c+5)\mathsf{M}(n) + o(\mathsf{M}(n))$$

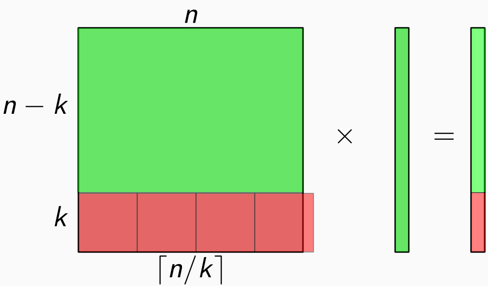
In-place middle product



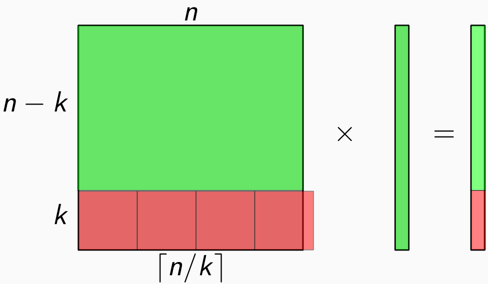
In-place middle product



In-place middle product

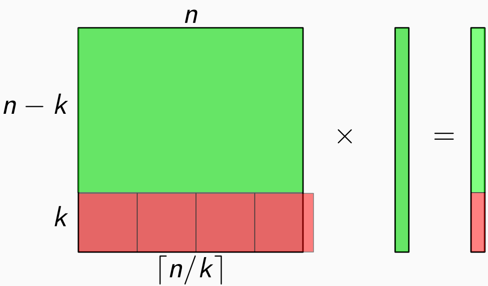


In-place middle product



- Recursive call on chunks of f... but with full g!
- $T(n,m) = \lceil n/k \rceil \mathrm{M}(k) + T(n, m-k)$

In-place middle product



- Recursive call on chunks of f... but with full g!
- $T(n,m) = \lceil {}^{n}/_{k} \rceil \mathsf{M}(k) + T(n, m-k)$

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

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!

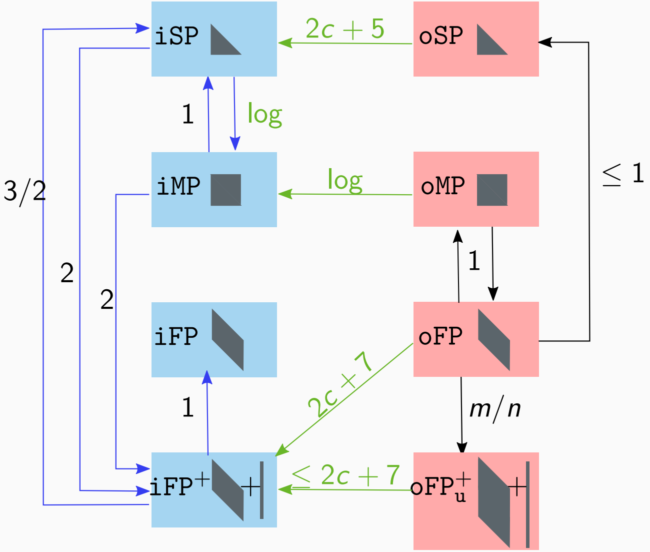
- Use our in-place algorithms as building blocks
  - Newton iteration: division, square root, ...
  - Evaluation & interpolation

→ (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 of the results



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!