# Space- and Time-Efficient Polynomial Multiplication

Daniel S. Roche

Symbolic Computation Group
School of Computer Science
University of Waterloo

ISSAC 2009
Seoul, Korea
30 July 2009

# Univariate Polynomial Multiplication

It’s important!

- Close cousin to integer multiplication
- Underlies many, many algorithms
- High-performance libraries developed and widely used
- Non-trivial algorithms useful in practice

# Specifics

## The Problem

Given: A ring R, an integer $n$,
and $f, g \in \mathsf{R}[x]$ with degrees less than $n$

Compute: Their product $f \cdot g \in \mathsf{R}[x]$

## The Model

- Ring operations have unit cost
- Random reads from input, random reads/writes to output
- Count size of auxiliary storage

# Univariate Multiplication Algorithms

| | Time Complexity | Space Complexity |
|---|---|---|
| **Classical Method** | $O(n^2)$ | $O(1)$ |
| **Divide-and-Conquer**<br>Karatsuba/Ofman ’63 | $O(n^{\log_2 3})$ or $O(n^{1.59})$ | $O(n)$ |
| **FFT-based**<br>Schönhage/Strassen ’71<br>Cantor/Kaltofen ’91 | $O(n \log n \log \log n)$ | $O(n)$ |

# Univariate Multiplication Algorithms

| | **Time Complexity** | **Space Complexity** |
|---|---|---|
| **Classical Method** | $O(n^2)$ | $O(1)$ |
| **Divide-and-Conquer**<br>Karatsuba/Ofman ’63 | $O(n^{\log_2 3})$ or $O(n^{1.59})$ | $O(n)$ |
| **FFT-based**<br>Schönhage/Strassen ’71<br>Cantor/Kaltofen ’91 | $O(n \log n \log \log n)$ | $O(n)$ |

Goal: Keep time complexity the same, reduce space

# Previous Work

- Savage & Swamy ’79; Abrahamson ’85
  $\Omega(n^2)$ lower bound for time $\times$ space under restrictive models
- Maeder 1993: Bounds extra space for Karatsuba multiplication so that storage can be preallocated
  — about $2n$ extra memory cells required.
- Thomé 2002: Karatsuba multiplication for polynomials using $n$ extra memory cells.

# Present Contributions

- New Karatsuba-like algorithm with $O(\log n)$ space
- New FFT-based algorithm with $O(1)$ space
  *under certain conditions*
- Implementations in C over $\mathbb{Z}/p\mathbb{Z}$

# Standard Karatsuba Algorithm

Idea: Reduce one degree-$2k$ multiplication to three of degree $k$.

- Originally noticed by Gauss (multiplying complex numbers), rediscovered and formalized by Karatsuba & Ofman

**Input**: $f, g \in \mathrm{R}[x]$ each with degree less than $2k$.

Write $f = f_0 + f_1 x^k$ and $g = g_0 + g_1 x^k$.

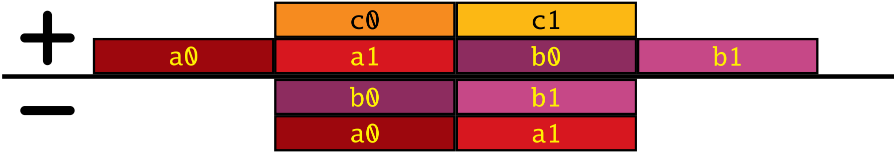

**Compute**: $a = f_0 g_0, \qquad b = f_1 g_1, \qquad c = (f_0 + f_1)(g_0 + g_1)$

$$f \cdot g = a + (c - a - b)x^k + bx^{2k}$$

# Low-Space Karatsuba Algorithm

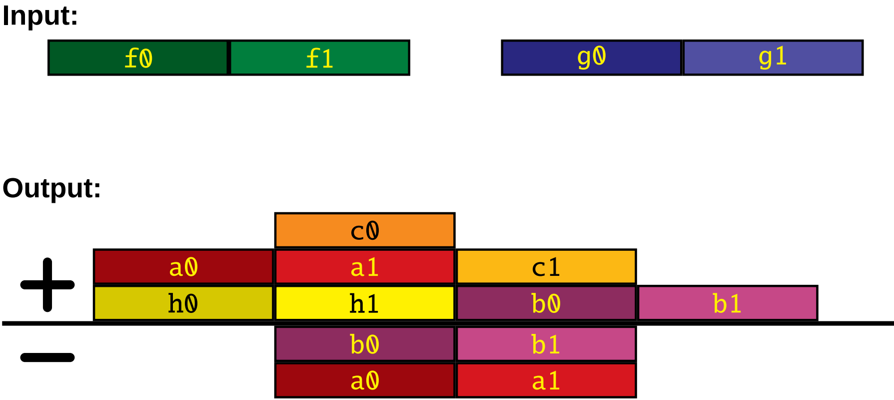
**Input:**


**Output:**



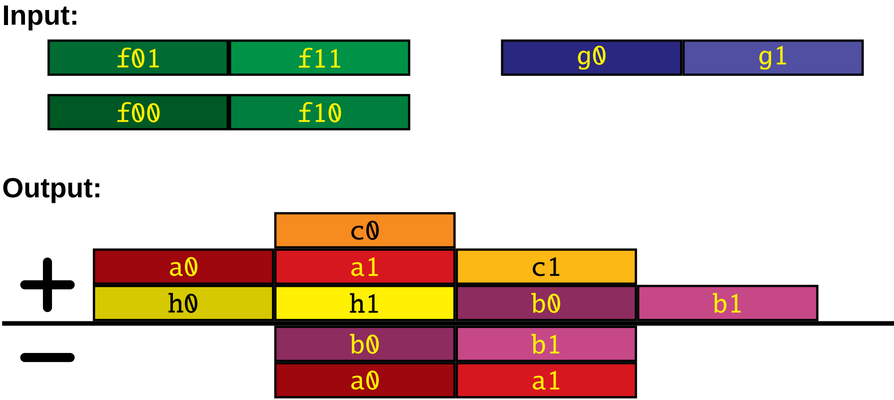
# Low-Space Karatsuba Algorithm

1. The low-order coefficients of the output are initialized as $h$, and the product $f \cdot g$ is added to this.



# Low-Space Karatsuba Algorithm

1. The low-order coefficients of the output are initialized as $h$, and the product $f \cdot g$ is added to this.
2. The first polynomial $f$ is given as a sum $f^{(0)} + f^{(1)}$.



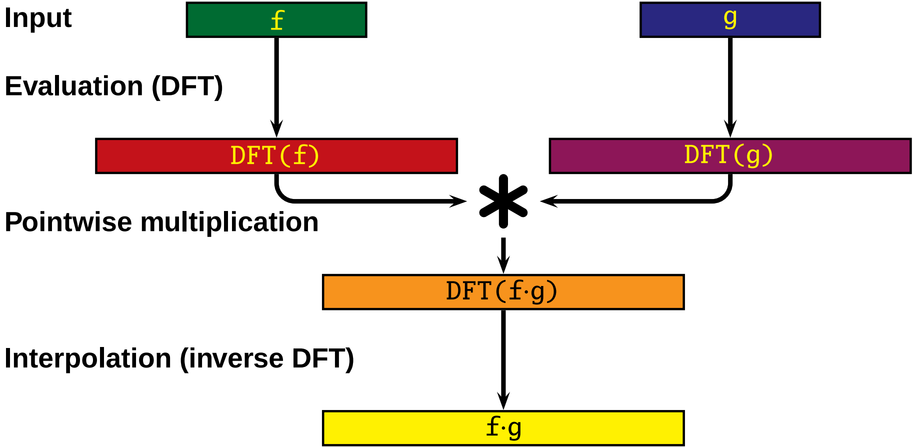
# A few details

Slight modifications are needed to handle all cases:

- Initial calls without extra conditions
- Operands with odd sizes
- Operands with different sizes

Result: First algorithm with $o(n^2)$ time $\times$ space

# DFT-Based Multiplication



# Primitive Roots of Unity

## Assumption

- $\deg f + \deg g < n = 2^k$ for some $k \in \mathbb{N}$
- The base ring R contains a $2^k$-PRU $\omega$

That is, assume “virtual roots of unity” have already been added; we will optimize from there.

# Folded Polynomials

Recall that $n = 2^k$ is the size of the output.

## Definition (Folded Polynomials)

$$f_i = f(\omega^{2^{i-1}} x) \quad \text{rem}\, x^{2^{k-i}} - 1$$

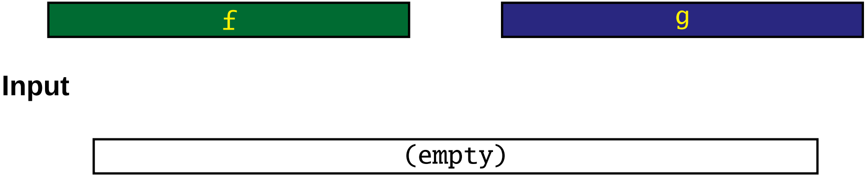
## Theorem

$$f\left(\omega^{2^i(2j+1)}\right) = f_{i+1}\left(\omega^{2^{i+1}j}\right)$$

So by computing each $f_i$ at all powers of $\omega^i$,
we get the values of $f$ at all powers of $\omega$.

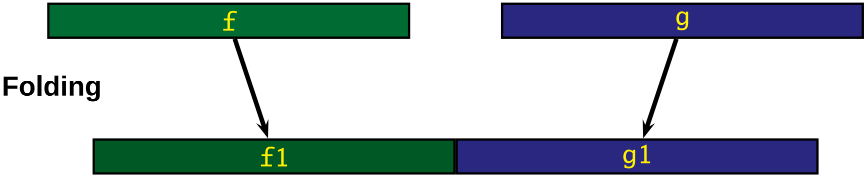
# FFT-Based Multiplication without Extra Space

**Idea**: Solve half of remaining problem at each iteration



# FFT-Based Multiplication without Extra Space

**Idea**: Solve half of remaining problem at each iteration



# FFT-Based Multiplication without Extra Space

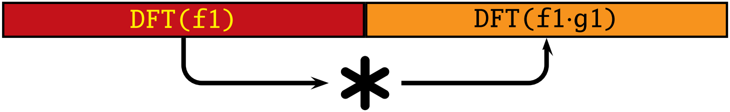
**Idea**: Solve half of remaining problem at each iteration


**In-Place FFTs (alternate formulation)**

# FFT-Based Multiplication without Extra Space

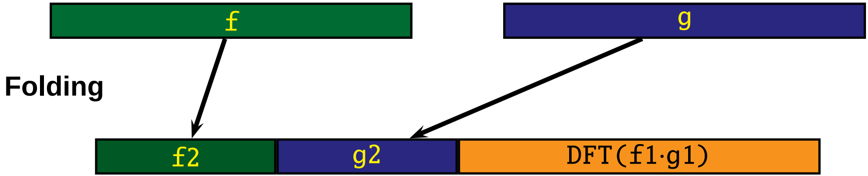
**Idea**: Solve half of remaining problem at each iteration


**Pointwise Multiplication**



# FFT-Based Multiplication without Extra Space

**Idea**: Solve half of remaining problem at each iteration



# FFT-Based Multiplication without Extra Space

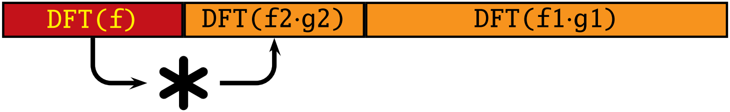
**Idea**: Solve half of remaining problem at each iteration


**In-Place FFTs (alternate formulation)**

# FFT-Based Multiplication without Extra Space

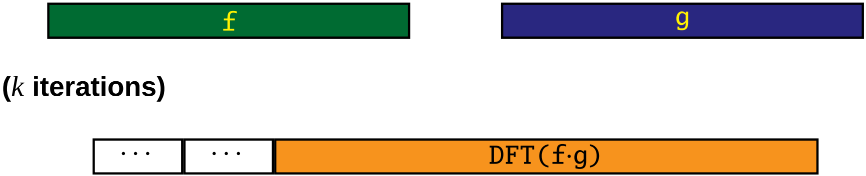
**Idea**: Solve half of remaining problem at each iteration


**Pointwise Multiplication**



# FFT-Based Multiplication without Extra Space

**Idea**: Solve half of remaining problem at each iteration

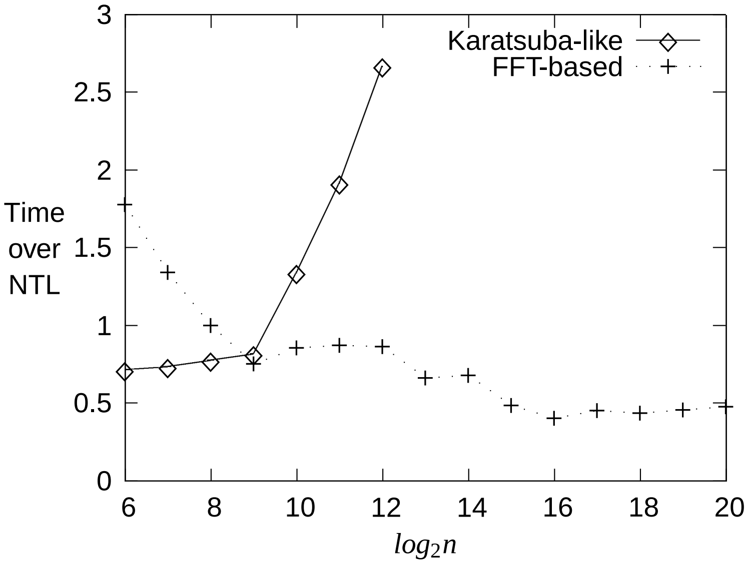


# FFT-Based Multiplication without Extra Space

**Idea**: Solve half of remaining problem at each iteration


**In-Place Reverse FFT (usual formulation)**

# Timing Benchmarks



# Future Directions

- Efficient implementation over $\mathbb{Z}$ (GMP)
- Similar results for
  Toom-Cook 3-way or $k$-way
- Parallelism!
- Is completely in-place (overwriting input) possible?