Between Dense and Sparse Polynomial Multiplication

Daniel S. Roche

WATERLOO
CHERITON SCHOOL OF COMPUTER SCIENCE

Drexel University
May 9, 2011

What is a polynomial?

A polynomial is any formula involving $+, -, \times$ on indeterminates and constants from a ring R.

Examples over $R = \mathbb{Z}$

$$4x^{10} - 3x^8 - x^7 + 3x^6 + x^5 - 2x^4 + 2x^3 + 5x^2$$

What is a polynomial?

A polynomial is any formula involving $+, -, \times$ on indeterminates and constants from a ring R.

Examples over $R = \mathbb{Z}$

$$4x^{10} - 3x^8 - x^7 + 3x^6 + x^5 - 2x^4 + 2x^3 + 5x^2$$

$$x^{451} - 9x^{324} - 3x^{306} + 9x^{299} + 4x^{196} - 9x^{155} - 2x^{144} + 10x^{27}$$

What is a polynomial?

A polynomial is any formula involving $+, -, \times$ on indeterminates and constants from a ring R.

Examples over R = $\mathbb{Z}$

$$4x^{10} - 3x^8 - x^7 + 3x^6 + x^5 - 2x^4 + 2x^3 + 5x^2$$

$$x^{451} - 9x^{324} - 3x^{306} + 9x^{299} + 4x^{196} - 9x^{155} - 2x^{144} + 10x^{27}$$

$$6x^{484} - 9x^{482} + 10x^{481} - 2x^{33} - 2x^{32} - 7x^{29} + 8x^{28} - 7x^{27}$$

What is a polynomial?

A polynomial is any formula involving $+, -, \times$ on indeterminates and constants from a ring R.

Examples over R = $\mathbb{Z}$

$$4x^{10} - 3x^8 - x^7 + 3x^6 + x^5 - 2x^4 + 2x^3 + 5x^2$$

$$x^{451} - 9x^{324} - 3x^{306} + 9x^{299} + 4x^{196} - 9x^{155} - 2x^{144} + 10x^{27}$$

$$6x^{484} - 9x^{482} + 10x^{481} - 2x^{33} - 2x^{32} - 7x^{29} + 8x^{28} - 7x^{27}$$

$$-x^{426} - 6x^{273} + 10x^{246} - 10x^{210} + 2x^{156} - 9x^{48} - 3x^{21} - 9x^{12}$$

Polynomial Arithmetic

- Addition/subtraction of polynomials is trivial.
- Division uses multiplication as a subroutine.
- Multiplication is the most important basic computational problem on polynomials.

Application areas

- Cryptography
- Coding theory
- Symbolic computation
- Scientific computing
- Experimental mathematics

Polynomial Representations

Let $f = 7 + 5xy^8 + 2x^6y^2 + 6x^6y^5 + x^{10}$.

Dense representation:

0	5	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	6	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1

Degree d
n variables
t nonzero terms

Dense size:
$O(d^n)$ coefficients

Polynomial Representations

Let $f = 7 + 5xy^8 + 2x^6y^2 + 6x^6y^5 + x^{10}$.

Recursive dense representation:

0	5					0				0
0	0					0				0
0	0					0				0
0	0					6				0
0	0					0				0
0	0					0				0
0	0					2				0
0	0					0				0
7	0	0	0	0	0	0	0	0	0	1

Degree d
n variables
t nonzero terms

Recursive dense size:
$O(tdn)$ coefficients

Polynomial Representations

Let $f = 7 + 5xy^8 + 2x^6y^2 + 6x^6y^5 + x^{10}$.

Sparse representation:

Degree d
n variables
t nonzero terms

Sparse size:
$O(t)$ coefficients
$O(tn \log d)$ bits

```
0 8 2 5 0
0 1 6 6 10
7 5 2 6 1
```

Dense Multiplication Algorithms

Cost (in ring operations) of multiplying two univariate dense polynomials with degrees less than d:

	Cost
Classical Method	$O(d^2)$
Divide-and-Conquer Karatsuba '63	$O(d^{\log_2 3})$ or $O(d^{1.59})$
FFT-based Schönhage/Strassen '71 Cantor/Kaltofen '91	$O(d \log d \operatorname{llog} d)$

We write $\mathsf{M}(d)$ for this cost.

Sparse Multiplication Algorithms

Cost of multiplying two univariate sparse polynomials with degrees less than d and at most t nonzero terms:

	Ring operations	**Bit operations**
Naïve	t^2	$O(t^3 \log d)$
Geobuckets (Yan '98)	t^2	$O(t^2 \log t \log d)$
Heaps (Johnson '74) (Monagan & Pearce '07)	t^2	$O(t^2 \log t \log d)$

Application: Cryptography

Public key cryptography is used extensively in communications. There are two popular flavors:

RSA

Requires integer computations modulo a large integer (thousands of bits). Long integer multiplication algorithms are generally the same as those for (dense) polynomials.

ECC

Usually requires computations in a finite extension field — i.e. computations modulo a polynomial (degree in the hundreds).

In both cases, sparse integers/polynomials are used to make schemes more efficient.

Application: Nonlinear Systems

Nonlinear systems of polynomial equations can be used to describe and model a variety of physical phenomena.

Numerous methods can be used to solve nonlinear systems, but usually:

- Inputs are sparse multivariate polynomials
- Intermediate values become dense.

One approach (used in triangular sets) simply switches from sparse to dense methods heuristically.

Adaptive multiplication

Goal: Develop algorithms which smoothly interpolate the cost between existing dense and sparse methods.

Ground rules

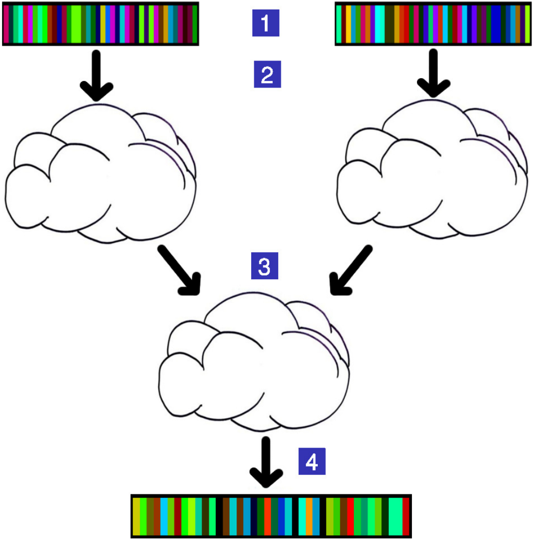
1. The cost must never be greater than any standard dense or sparse algorithm.
2. The cost should be less than both in many “easy cases”.

Overall Approach

Overall Steps

1. Recognize structure
2. Change rep. to exploit structure
3. Multiply
4. Convert back

- Step 3 cost depends on instance difficulty.
- Steps 1, 2, 4 must be *fast* (linear time).



Chunky Polynomials

Example

- $f = 5x^6 + 6x^7 - 4x^9 - 7x^{52} + 4x^{53} + 3x^{76} + x^{78}$

Chunky Polynomials

Example

- $f = 5x^6 + 6x^7 - 4x^9 - 7x^{52} + 4x^{53} + 3x^{76} + x^{78}$
- $f_1 = 5 + 6x - 4x^3, \qquad f_2 = -7 + 4x, \qquad f_3 = 3 + x^2$
- $f = f_1 x^6 + f_2 x^{52} + f_3 x^{76}$

Chunky Multiplication

Sparse algorithms on the outside, *dense* algorithms on the inside.

- Exponent arithmetic stays the same.
- Coefficient arithmetic is more costly.
- Terms in product may have more overlap.

Theorem

Given

$$\begin{aligned} f &= f_1 x^{e_1} + f_2 x^{e_2} + \cdots + f_t x^{e_t} \\ g &= g_1 x^{d_1} + g_2 x^{d_2} + \cdots + g_s x^{d_s}, \end{aligned}$$

the cost of chunky multiplication (in ring operations) is

$$O\left(\sum_{\substack{\deg f_i \geq \deg g_j \\ 1 \leq i \leq t,\ 1 \leq j \leq s}} (\deg f_i) \cdot \mathsf{M}\left(\frac{\deg g_j}{\deg f_i}\right) + \sum_{\substack{\deg f_i < \deg g_j \\ 1 \leq i \leq t,\ 1 \leq j \leq s}} (\deg g_j) \cdot \mathsf{M}\left(\frac{\deg f_i}{\deg g_j}\right) \right).$$

Conversion to the Chunky Representation

Initial idea: Convert each operand independently,
then multiply in the chunky representation.

But how to minimize the nasty cost measure?

Theorem

*Any **independent** conversion algorithm must result in slower multiplication than the dense or sparse method in some cases.*

Two-Step Chunky Conversion

First Step: Optimal Chunk Size

Suppose every chunk in both operands was forced to have the same size k.

This simplifies the cost to $t(k) \cdot s(k) \cdot \mathsf{M}(k)$,
where $t(k)$ is the least number of size-k chunks to make f.

First conversion step:
Compute the optimal value of k to minimize this simplified cost measure.

Computing the optimal chunk size

Optimal chunk size computation algorithm:

1. Create two min-heaps with all “gaps” in f and g, ordered on the size of resulting chunk if gap were removed.
2. Remove all gaps of smallest priority and update neighbors
3. Approximate $t(k), s(k)$ by the size of the heaps, and compute $t(k) \cdot s(k) \cdot \mathsf{M}(k)$
4. Repeat until no gaps remain.

With careful implementations, this can be made linear-time in either the dense or sparse representation.
Constant factor approximation; ratio is 4.

Computing the optimal chunk size

Optimal chunk size computation algorithm:

1. Create two min-heaps with all “gaps” in f and g, ordered on the size of resulting chunk if gap were removed.
2. Remove all gaps of smallest priority and update neighbors
3. Approximate $t(k), s(k)$ by the size of the heaps, and compute $t(k) \cdot s(k) \cdot \mathsf{M}(k)$
4. Repeat until no gaps remain.

With careful implementations, this can be made linear-time in either the dense or sparse representation.
Constant factor approximation; ratio is 4.

Observe: We must compute the cost of dense multiplication!

Two-Step Chunky Conversion

Second Step: Conversion given chunk size

After computing "optimal chunk size", conversion proceeds independently.

We compute the optimal chunky representation for multiplying by a single size-k chunk.

Idea: For each gap, maintain a linked list of all previous gaps to include if the polynomial were truncated here.

Algorithm: Increment through gaps, each time finding the last gap that should be included.

Conversion given optimal chunk size

The algorithm uses two key properties:

- Chunks larger than k bring no benefit.
- For smaller chunks, we want to minimize $\sum_i \frac{\mathrm{M}(\deg f_i)}{\deg f_i}$.

Theorem

Our algorithm computes the optimal chunky representation for multiplying by a single size-k chunk.

Its cost is linear in the dense or sparse representation size.

Chunky Multiplication Overview

Input: $f, g \in \mathsf{R}[x]$, either in the sparse or dense representation
Output: Their product $f \cdot g$, in the same representation

1. Compute approximation to "optimal chunk size" k, looking at both f and g simultaneously.
2. Convert f to optimal chunky representation for multiplying by a single size-k chunk.
3. Convert g to optimal chunky representation for multiplying by a single size-k chunk.
4. Multiply pairwise chunks using dense multiplication.
5. Combine terms and write out the product.

Second idea for Adaptive Multiplication

Example

- $f = 3 - 2x^3 + 7x^6 + 5x^{12} - 6x^{15}$

Second idea for Adaptive Multiplication

Example

- $f = 3 - 2x^3 + 7x^6 + 5x^{12} - 6x^{15}$
- $f_D = 3 - 2x + 7x^2 + 5x^4 - 6x^5$
- $f = f_D \circ x^3$

Second idea for Adaptive Multiplication

Example

- $f = 3 - 2x^3 + 7x^6 + 5x^{12} - 6x^{15}$
- $f_D = 3 - 2x + 7x^2 + 5x^4 - 6x^5$
- $f = f_D \circ x^3$
- $g = g_D \circ x^3$

To multiply $f \cdot g$, multiply $f_D \cdot g_D$:

$$f \cdot g = (f_D \cdot g_D) \circ x^3$$

Different Spacing

Example

- $f = 4 + 6x^2 + 9x^4 - 7x^6 - x^8 + 3x^{10} - 2x^{12}$
- $g = 3 + 2x^3 - x^6 + 8x^9 - 5x^{12}$

Different Spacing

Example

- $f = 4 + 6x^2 + 9x^4 - 7x^6 - x^8 + 3x^{10} - 2x^{12}$
- $f_D = 4 + 6x + 9x^2 - 7x^3 - x^4 + 3x^5 - 2x^6$
- $f = f_D \circ x^2$

- $g = 3 + 2x^3 - x^6 + 8x^9 - 5x^{12}$
- $g_D = 3 + 2x - x^2 + 8x^3 - 5x^4$
- $g = g_D \circ x^3$

Different Spacing

Example

- $f = 4 + 6x^2 + 9x^4 - 7x^6 - x^8 + 3x^{10} - 2x^{12}$
- $f_0 = 4 - 7x - 2x^2,\qquad f_2 = 6 - x,\qquad f_4 = 9 + 3x$
- $f = f_0 \circ x^6 + x^2(f_2 \circ x^6) + x^4(f_4 \circ x^6)$

- $g = 3 + 2x^3 - x^6 + 8x^9 - 5x^{12}$
- $g_0 = 3 - x - 5x^2,\qquad g_3 = 2 + 8x$
- $g = g_0 \circ x^6 + x^3(g_3 \circ x^6)$

Different Spacing

Example

- $f = 4 + 6x^2 + 9x^4 - 7x^6 - x^8 + 3x^{10} - 2x^{12}$
- $f_0 = 4 - 7x - 2x^2, \qquad f_2 = 6 - x, \qquad f_4 = 9 + 3x$
- $f = f_0 \circ x^6 + x^2(f_2 \circ x^6) + x^4(f_4 \circ x^6)$

- $g = 3 + 2x^3 - x^6 + 8x^9 - 5x^{12}$
- $g_0 = 3 - x - 5x^2, \qquad g_3 = 2 + 8x$
- $g = g_0 \circ x^6 + x^3(g_3 \circ x^6)$

- Computing $f \cdot g$ requires 6 multiplications $f_i \cdot g_j$, *no additions*
- Note: $f \cdot g$ is almost totally dense.

Equal-Spaced Multiplication

Theorem

Given $f = f_D \circ x^k$, $g = g_D \circ x^\ell$, *and* $\deg f, \deg g < n$, *can find* $f \cdot g$ *using*

$$O\left(\frac{n}{\gcd(k,\ell)} \mathsf{M}\left(\frac{n}{\operatorname{lcm}(k,\ell)}\right)\right)$$

ring operations.

Allowing Outliers

Consider $f = 3 - 2x^3 + 7x^6 - 4x^7 + 5x^{12} - 6x^{15}$.

- Can we handle *almost*-equal spaced polynomials?

Allowing Outliers

Consider $f = 3 - 2x^3 + 7x^6 - 4x^7 + 5x^{12} - 6x^{15}$.

- Can we handle *almost*-equal spaced polynomials?

Idea: Write $f = x^r \cdot f_D(x^k) + f_S$, where f_S is s-sparse.

If $s \in O(\log \deg f)$, the cost of equal-spaced multiplication will be less than that of dense multiplication.

Equal-Spaced Conversion

Write $S = \{e_1, e_2, \ldots, e_t\}$ for the exponents of nonzero terms in f.

Goal: Find the largest k such that all but $\lg n$ of the e_i's are equivalent modulo k.

Equal-Spaced Conversion

Write $S = \{e_1, e_2, \ldots, e_t\}$ for the exponents of nonzero terms in f.

Goal: Find the largest k such that all but $\lg n$ of the e_i's are equivalent modulo k.

Problem: This is related to max-factor k-gcd, which is **NP**-hard.

- Eliminates linear-time optimal conversion from sparse.
- Is the problem easier for dense polynomials?

Conversion from Dense

Theorem (Upper bound on k)

$$k \in O\left(\frac{n}{t}\right).$$

Idea: Perform a check for each possible k.

- Single check requires t modular computations.
- Can find majority element in $O(t)$ by the algorithm of (Boyer & Moore / Fischer & Salzburg).
- Total cost is $O(tk)$, which is $O(n)$.

So we can find the optimal k in linear time
in the size of the dense representation.

Combining Chunky and Equal-Spaced

To avoid the need for converting from sparse to equal-spaced, we can combine with chunky polynomials.

Chunks with Equal Spacing:

1. Convert dense or sparse input to chunky representation.
2. Simultaneously convert each dense chunk to equal-spaced, using the same spacing parameter k for all chunks.
3. Multiplication now uses sparse multiplication on the outside, equal-spaced in the middle, and dense on the inside.

Theorem

Multiplication using chunks with equal spacing as above is never more costly than chunky or equal-spaced multiplication alone.

Implementations in a Software Library

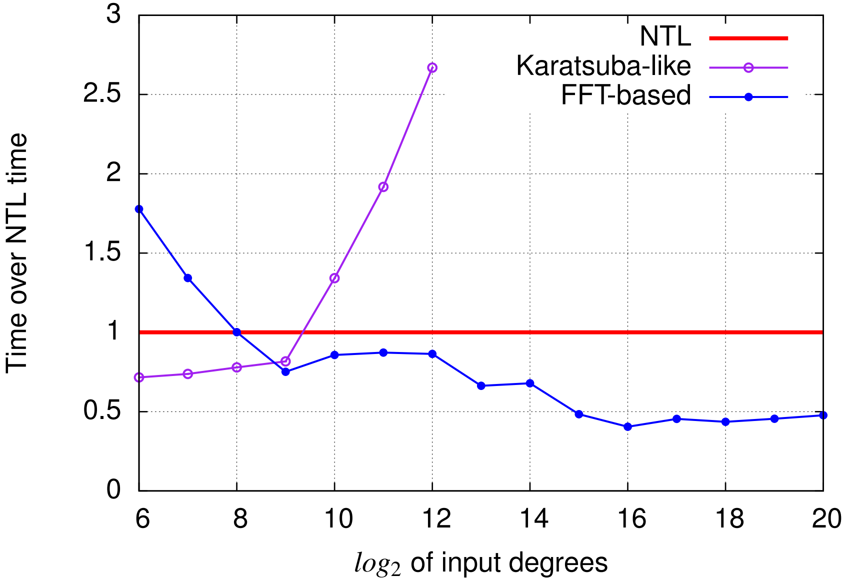
Our algorithms should be implemented and compared to existing approaches.

Current software supporting polynomial arithmetic is either:

- **Too big**: Maple, Mathematica, Singular, Magma, etc.
- **Too small**: NTL, FLINT, zn_poly
- **Not open**: sdmp, TRIP

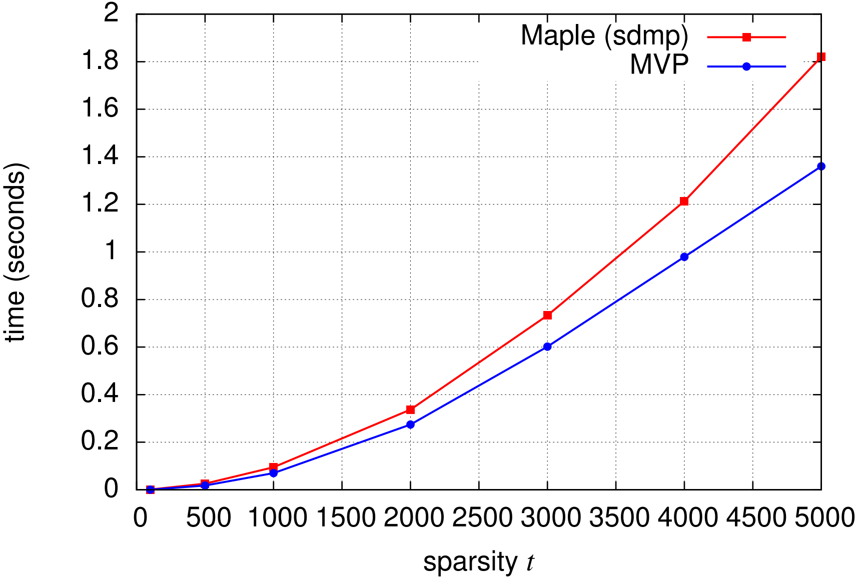
The MVP (MultiVariate Polynomials) library will fill a niche:
An open-source library for high-performance computations with sparse and dense polynomials.

Dense multiplication in MVP



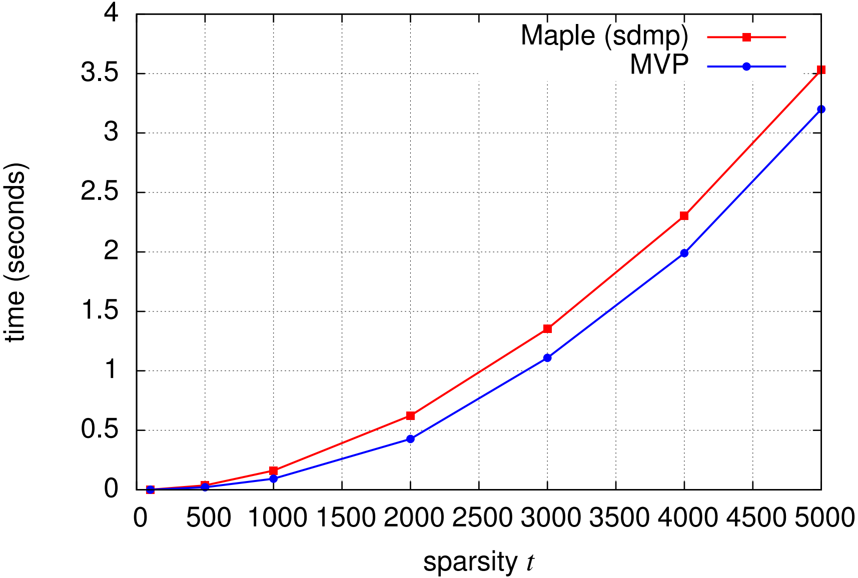
Sparse multiplication in MVP

Degree 10 000

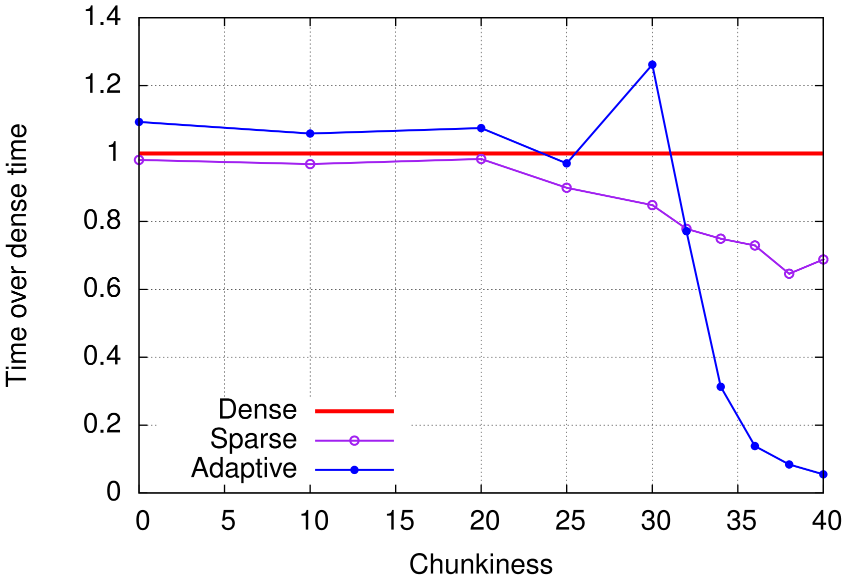


Sparse multiplication in MVP

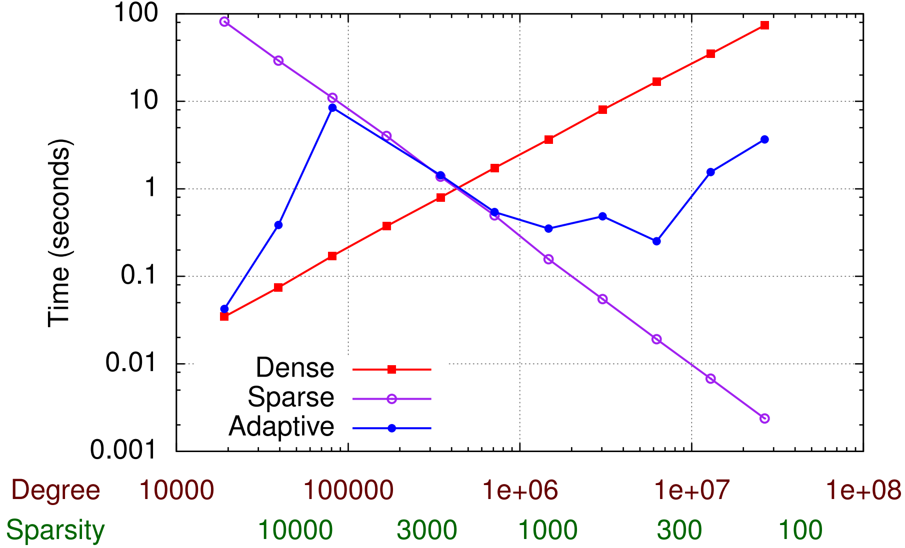
Degree 1 000 000



Timings vs “Chunkiness”



Timings without imposed chunkiness



Multivariate Multiplication

We can apply our univariate algorithms to the multivariate case with the **Kronecker substitution**:

Given multivariate polynomials $f, g \in \mathsf{R}[x_1, x_2, \ldots, x_n]$, we compute degree bounds on the product: $d_i > \deg_{x_i}(fg)$, then write

$$\hat{f} = f\left(x, x^{d_1}, x^{d_1 d_2}, \ldots, x^{d_1 d_2 \cdots d_{n-1}}\right)$$
$$\hat{g} = g\left(x, x^{d_1}, x^{d_1 d_2}, \ldots, x^{d_1 d_2 \cdots d_{n-1}}\right)$$

Computing the univariate product $\hat{f} \cdot \hat{g}$ then gives all the terms of the actual multivariate product.

Possible Applicability of Kronecker Substitution

Homogeneous Polynomials

- Every term in a *homogeneous polynomial* has the same total degree.
- Example: $f = 2xy^4z + x^2y^4 - 2x^3z^3 + x^4yz$
- In the Kronecker substitution, this polynomial will be equal-spaced.

Recursive Dense

- The *recursive dense* representation has shown to be generally useful in computer algebra systems (Stoutemyer 1984, Fateman 2003).
- A dense coefficient array in this representation corresponds to a chunk with equal spacing under the Kronecker substitution.

Hence our adaptive representations may work well for “real” data.

Better approximations and bounds

We can hope for better conversion algorithms:

- Chunky conversion steps are optimal, but overall process might not be.
- Chunky/equal-spaced combination might be sub-optimal.
- Better constant-factor approximation

On a practical level, further tweaking might avoid inefficiencies, especially at borderline cases.

The Ultimate Goal?

Adaptive methods go between pseudo-linear-time dense algorithms and quadratic-time sparse algorithms.

- Output size of sparse multiplication is at most quadratic.
- **Can we have pseudo-linear output-sensitive cost?**

Note: This is the best we can hope for and would generalize chunky and equal-spaced.

Connection to Sparse Interpolation

Problem: Sparse Polynomial Interpolation

Given a way to evaluate an unknown $f \in \mathsf{R}[x]$ at any point, determine the terms in f.

Ben-Or & Tiwari ('88): Sparse interpolation algorithm
Kaltofen & Lee ('02): Early termination strategy

These results allow sparse multiplication in time $\tilde{O}(t \log^2 d)$, where t is sparsity of input plus output — *almost* what we want.

Extra $O(\log d)$ factor comes from root finding and discrete logs — can these be avoided without increasing the cost?

Connection to Sparse Interpolation

Problem: Sparse Polynomial Interpolation

Given a way to evaluate an unknown $f \in \mathsf{R}[x]$ at any point, determine the terms in f.

Ben-Or & Tiwari ('88): Sparse interpolation algorithm
Kaltofen & Lee ('02): Early termination strategy

These results allow sparse multiplication in time $\tilde{O}(t \log^2 d)$, where t is sparsity of input plus output — *almost* what we want.

Extra $O(\log d)$ factor comes from root finding and discrete logs — can these be avoided without increasing the cost?

Are multiplication and interpolation deeply linked?

Summary

- Two ways to go between dense and sparse multiplication: chunky and equal-spaced multiplication.
- These algorithms are never worse than existing approaches, but can be much better in well-structured cases.
- The two ideas can be combined and also have some promising applications and implementation results.
- Much work remains to understand this fundamental problem.

Thank you!