Between Sparse and Dense Arithmetic

Daniel S. Roche

Computer Science Department
United States Naval Academy

NARC Seminar
November 28, 2012

The Problem

People want to compute with really big numbers and polynomials.

Two basic choices for representation:

- **Dense** — wasted space, but fast algorithms
- **Sparse** — compact storage, slower algorithms

The goal: Alternative representations and algorithms that go smoothly between these two options

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.

Current Focus: Polynomial Multiplication

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

More application areas

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

What is a polynomial?

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

Examples with integer coefficients (R = $\mathbb{Z}$)

$$x^{10} + x^9 + x^8 + x^7 + x^6 + 1$$

What is a polynomial?

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

Examples with integer coefficients (R = $\mathbb{Z}$)

$$x^{10} + x^9 + x^8 + x^7 + x^6 + 1$$

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

What is a polynomial?

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

Examples with integer coefficients (R = $\mathbb{Z}$)

$$x^{10} + x^9 + x^8 + x^7 + x^6 + 1$$

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

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

What is a polynomial?

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

Examples with integer coefficients (R = $\mathbb{Z}$)

$$x^{10} + x^9 + x^8 + x^7 + x^6 + 1$$

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

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

$$x^{426} - 6x^{273}y^{399}z^2 + 10x^{246}yz^{201} - 10x^{210}y^{401} - 3x^{21}yz - 9z^{12}$$

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
```

Aside: Cost Measures

Measure of Success

The “cost” of an algorithm is measured as its rate of growth as the input size increases.

Most relevant costs:

- $O(n)$: Linear time
  Intractable when $n \geq 10^{12}$ or so.
- $O(n \log n)$: Linearithmic time
  Intractable when $n \geq 10^{10}$ or so.
- $O(n^2)$: Quadratic time
  Intractable when $n \geq 10^{6}$ or so.

Direct Multiplication

Most methods work by directly multiplying coefficients, adding them up, and so on.

Example: "School" Multiplication

$$\begin{array}{rr} & 332 \\ \times & 213 \\ \hline \end{array}$$

Direct Multiplication

Most methods work by directly multiplying coefficients, adding them up, and so on.

Example: “School” Multiplication

$$
\begin{array}{rr}
 & 332 \\
\times & 213 \\
\hline
 & 6
\end{array}
$$

Direct Multiplication

Most methods work by directly multiplying coefficients, adding them up, and so on.

Example: “School” Multiplication

$$
\begin{array}{lr}
 & 332 \\
\times & 213 \\
\hline
 & 96
\end{array}
$$

Direct Multiplication

Most methods work by directly multiplying coefficients, adding them up, and so on.

Example: "School" Multiplication

$$\begin{array}{rr} & 332 \\ \times & 213 \\ \hline & 996 \end{array}$$

Direct Multiplication

Most methods work by directly multiplying coefficients, adding them up, and so on.

Example: "School" Multiplication

$$
\begin{array}{rr}
 & 332 \\
\times & 213 \\
\hline
 & 996 \\
 & 332 \\
 & 664
\end{array}
$$

Direct Multiplication

Most methods work by directly multiplying coefficients, adding them up, and so on.

Example: "School" Multiplication

$$
\begin{array}{rr}
 & 332 \\
\times & 213 \\
\hline
 & 996 \\
 & 332 \\
+ & 664 \\
\hline
 & 70716
\end{array}
$$

Total cost:
$O(n^2)$
(quadratic)

Indirect Multiplication

Some faster methods do their work in an alternate representation:

1. Convert input polynomials to alternate representation
2. Multiply in the alternate representation
3. Convert the product back to the original form

FFT-Based Multiplication mod 5

$$f = 2x + 3 \qquad g = x^2 + 2x + 3$$

Indirect Multiplication

Some faster methods do their work in an alternate representation:

1. Convert input polynomials to alternate representation
2. Multiply in the alternate representation
3. Convert the product back to the original form

FFT-Based Multiplication mod 5

$$f = 2x + 3 \qquad g = x^2 + 2x + 3$$

1. Evaluate each polynomial at $x = 1, 3, 4, 2$:

$$f_{\mathrm{alt}} = [0, 4, 1, 2] \qquad g_{\mathrm{alt}} = [1, 3, 2, 1]$$

Indirect Multiplication

Some faster methods do their work in an alternate representation:

1. Convert input polynomials to alternate representation
2. Multiply in the alternate representation
3. Convert the product back to the original form

FFT-Based Multiplication mod 5

$$f = 2x + 3 \qquad g = x^2 + 2x + 3$$

1. Evaluate each polynomial at $x = 1, 3, 4, 2$:

$$f_{\text{alt}} = [0, 4, 1, 2] \qquad g_{\text{alt}} = [1, 3, 2, 1]$$

2. Multiply the evaluations pairwise:

$$(fg)_{\text{alt}} = [0, 2, 2, 2]$$

Indirect Multiplication

Some faster methods do their work in an alternate representation:

1. Convert input polynomials to alternate representation
2. Multiply in the alternate representation
3. Convert the product back to the original form

FFT-Based Multiplication mod 5

$$f = 2x + 3 \qquad g = x^2 + 2x + 3$$

1. Evaluate each polynomial at $x = 1, 3, 4, 2$:

$$f_{\text{alt}} = [0, 4, 1, 2] \qquad g_{\text{alt}} = [1, 3, 2, 1]$$

2. Multiply the evaluations pairwise:

$$(fg)_{\text{alt}} = [0, 2, 2, 2]$$

3. Interpolate at $x = 1, 3, 4, 2$:

$$fg = 2x^3 + 2x^2 + 2x + 4$$

Dense Multiplication Algorithms

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

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

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:

	Cost	Method
Naïve	$O(t^3 \log d)$	Direct
Geobuckets (Yan '98)	$O(t^2 \log t \log d)$	Direct
Heaps (Johnson '74) (Monagan & Pearce '07)	$O(t^2 \log t \log d)$	Direct

Adaptive multiplication

Goal: Develop new algorithms whose cost smoothly varies 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”.

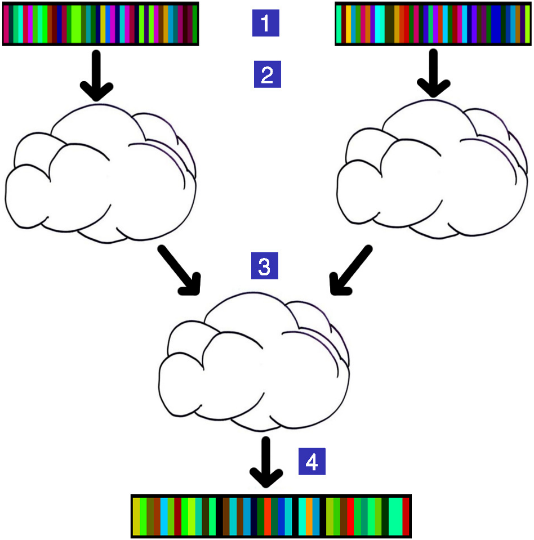
Primary technique: Develop indirect methods for the sparse case.

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

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

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.

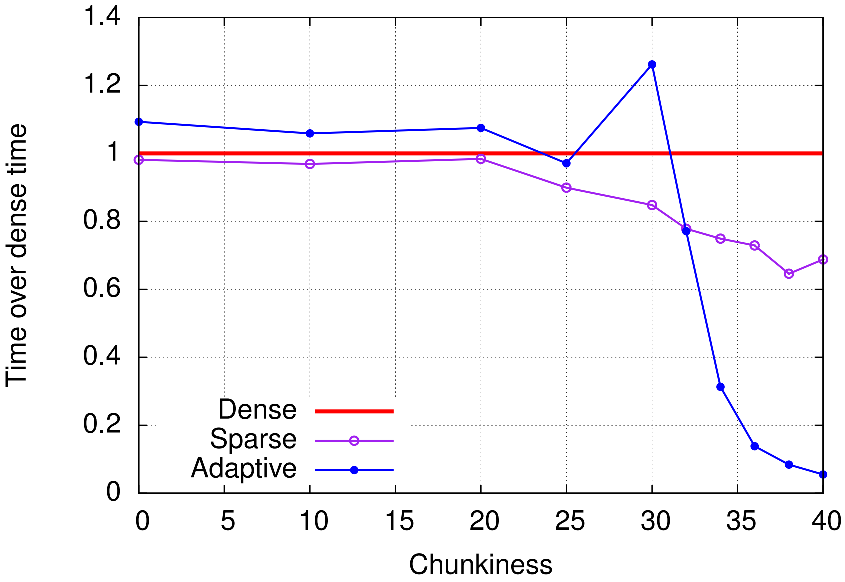
Its cost is linear in the dense or sparse representation size.

Chunky Multiplication Overview

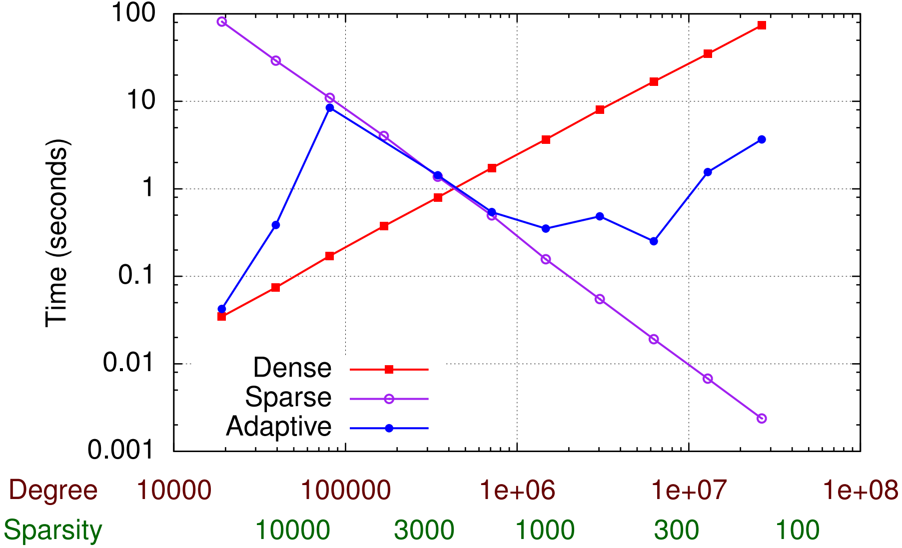
Input: $f, g \in \mathrm{R}[x]$, either in the sparse or dense representation
Output: Their product $f \cdot g$, in the same representation

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.

Timings vs "Chunkiness"



Timings without imposed chunkiness



The Ultimate Goal?

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

- Output size of dense multiplication is always linear.
- Output size of sparse multiplication is at most quadratic, but might be much less
- **Can we have linearithmic output-sensitive cost?**

Note: This is the best we can hope for and would generalize the "chunky" approach.

Summary

- New indirect multiplication methods to go between existing sparse and dense multiplication algorithms.
- Chunky multiplication is never (asymptotically) worse than existing approaches, but can be much better in well-structured cases.
- Recent results on sparse FFTs and sparse interpolation may lead to a significant breakthrough in theory.
- Much work remains to be done!