Adaptive Polynomial Multiplication

Daniel S. Roche

Symbolic Computation Group
School of Computer Science
University of Waterloo

ORCCA Joint Lab Meeting
University of Western Ontario
14 March 2008

Outline

How to Represent Univariate Polynomials

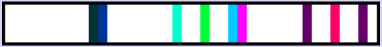
Let $f \in \mathrm{R}[x]$ with degree n, s nonzero terms.

Dense Representation

Write down every coefficient. Size is $O(n)$:

$$f = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Sparse Representation



Only write down nonzero terms. Size is $O(s \log n)$:

$$f = c_1 x^{e_1} + c_2 x^{e_2} + \cdots + c_s x^{e_s}$$

What about multivariate?

Multivariate Polynomial Representations

- Completely dense (size grows exponentially)
- Distributed sparse (default in Maple)
- Recursive dense (the best?)
- Variations on these. . .

- Essentially no different algorithms for multiplication
- Univariate algorithms generalize

Dense Multiplication Algorithms

Let R be an arbitrary ring, and $f, g \in \mathsf{R}[x]$.

Definition

$\mathsf{M}(n)$ is the number of operations in R to compute $h = f \cdot g$ with $\deg f, \deg g < n$.

- Classical: $\mathsf{M}(n) \in O(n^2)$
- Karatsuba & Ofman (1963): $\mathsf{M}(n) \in O(n^{\log_2 3})$
- Schönhage & Strassen (1971), Cantor & Kaltofen (1991): $\mathsf{M}(n) \in O(n \log n \log\log n)$ — uses FFT

If $\deg g < m \leq n$, can multiply $f \cdot g$ with $O(\frac{n}{m}\mathsf{M}(m))$.

Assumptions on $\mathsf{M}(n)$

- If R has a 2^k-PRU, with $2^k \geq 2n$, then $\mathsf{M}(n) \in O(n \log n)$.
- Under "bounded coefficients model", $\mathsf{M}(n) \in \Omega(n \log n)$. (Bürgisser & Lotz 2004)

So we (reasonably) assume $\mathsf{M}(n) \in \Theta(n \log n)$.
This will simplify the analysis.

Sparse Polynomial Multiplication

- Naïve: $O(s^2)$ ring ops:
  Optimal since $f \cdot g$ could have s^2 terms.
- Geobuckets (Yan 1998): Optimal bit complexity
- Heaps (Johnson 1974, Monagan & Pearce 2007):
  Optimal space complexity

Adaptive Sorting

List sorting is the birthplace of adaptive analysis (Melhorn 1984).

- Classical problem in computer science.
- Lower bound (comparisons) is $\Omega(n \log n)$.
- Maching upper bound algorithms (e.g. MergeSort)

Question: Can we do better on “almost” sorted lists?
Answer: Yes!

Adaptive sorting interesting theoretically *and* useful in practice.

A rose by any other name. . .

Related Notions

- Output-Sensitive Algorithms
- Early Termination
- Parameterized Complexity

These terms are not foreign to computer algebra!

Examples: Sparse interpolation, Chinese remaindering

General Approach to Adaptive Altorithms

Definition

An *adaptive algorithm* is one whose complexity
depends not only on the size of the input,
but also on some measure of difficulty.

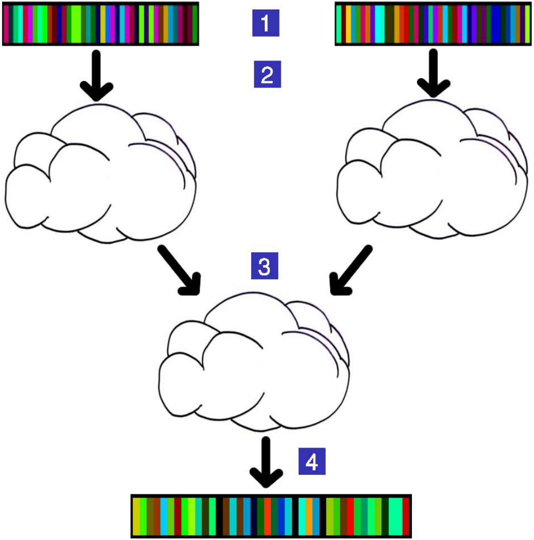
- Finer level of analysis
- Still require worst-case complexity not to be worse
- The goal: improvement in many “easy” cases.

Our 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).



An Obvious Adaptive Algorithm

Algorithm

1. Determine whether sparse or dense multiplication will be faster
2. (Possibly) convert to faster representation
3. Multiply using known methods
4. (Possibly) convert back

- Cost: $O\left(\min\left\{\mathrm{M}(n), s^2\right\}\right)$
- Has been suggested for triangular decomposition, where intermediate expressions can become dense.

Sequential Coefficients

Example

$$f = 1 + 2x + 3x^2 + 4x^3 + \cdots \qquad = \sum (i+1)x^i$$
$$g = -2 + 7x - 3x^2 - 4x^3 + \cdots \qquad \text{(arbitrary)}$$

Can compute $f \cdot g$ with an accumulator:

$$f \cdot g =$$
$$\texttt{accum} =$$

Sequential Coefficients

Example

$$f = 1 + 2x + 3x^2 + 4x^3 + \cdots \qquad = \sum (i+1)x^i$$
$$g = -2 + 7x - 3x^2 - 4x^3 + \cdots \qquad \text{(arbitrary)}$$

Can compute $f \cdot g$ with an accumulator:

$$f \cdot g = -2$$
$$\texttt{accum} = -2$$

Sequential Coefficients

Example

$$\begin{aligned} f &= 1 + 2x + 3x^2 + 4x^3 + \cdots & &= \sum (i+1)x^i \\ g &= -2 + 7x - 3x^2 - 4x^3 + \cdots & &\text{(arbitrary)} \end{aligned}$$

Can compute $f \cdot g$ with an accumulator:

$$\begin{aligned} f \cdot g &= -2 + 3x \\ \texttt{accum} &= 5 \end{aligned}$$

Sequential Coefficients

Example

$$f = 1 + 2x + 3x^2 + 4x^3 + \cdots \qquad = \sum (i+1)x^i$$
$$g = -2 + 7x - 3x^2 - 4x^3 + \cdots \qquad \text{(arbitrary)}$$

Can compute $f \cdot g$ with an accumulator:

$$f \cdot g = -2 + 3x + 5x^2$$
$$\texttt{accum} = 2$$

Sequential Coefficients

Example

$$
\begin{aligned}
f &= 1 + 2x + 3x^2 + 4x^3 + \cdots && = \sum (i+1)x^i \\
g &= -2 + 7x - 3x^2 - 4x^3 + \cdots && \text{(arbitrary)}
\end{aligned}
$$

Can compute $f \cdot g$ with an accumulator:

$$
\begin{aligned}
f \cdot g &= -2 + 3x + 5x^2 + 3x^3 \\
\texttt{accum} &= -2
\end{aligned}
$$

Sequential Multiplication

Works for any arithmetic-geometric sequence:

$$f = \sum_{i=0}^{n} (c_1 + c_2 i + c_3 c_4^i) x^i$$

For arbitrary $g \in \mathsf{R}[x]$,
can compute $f \cdot g$ in linear time.

- This is optimal!

Generalization

Split arbitrary $f \in \mathsf{R}[x]$ into:

$$f = f_S + f_N,$$

where

- f_S has sequential coefficients
- f_N (the “noise”) is very small

Can determine f_S by finding successive differences, quotients.

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.

- Again, allow for noise: $f = f_D \circ x^k + f_N$
- Finding optimal k value related to max factor gcd

Simple Marriage of Dense and Sparse

Idea: Sparse polynomials with dense polynomial coefficients.

Example

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

Simple Marriage of Dense and Sparse

Idea: Sparse polynomials with dense polynomial coefficients.

Example

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

Simple Marriage of Dense and Sparse

Idea: Sparse polynomials with dense polynomial coefficients.

Example

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

In general, write $f = f_1 x^{e_1} + f_2 x^{e_2} + \cdots + f_t x^{e_t}$

- $t = 1$: Dense representation
- $\deg f_i = 0$: Sparse representation

Chunky Multiplication Algorithm

Multiplication is sparse on outer loop, dense on inner loop

- Exploits sparsity *and* uses fast dense algorithms
- Can be faster than sparse and dense algorithms:

Example

$f, g \in \mathsf{R}[x]$ with $\deg f, \deg g < n$, and
f, g each have $\log_2 n$ dense chunks with degrees less than $\sqrt{n}$.

Costs (ring operations):

- Dense: $\mathsf{M}(n)$, or $\Omega(n \log n)$
- Sparse: $\Omega(n \log^2 n)$
- Chunky: $O(\sqrt{n} \log^3 n \operatorname{loglog} n)$

Limitations

Can't always be faster than both dense and sparse:

Example

$f, g \in \mathsf{R}[x]$, degrees $< n$, each with $\sqrt{n}$ nonzero terms, spaced equally apart.

- Dense, sparse multiplication cost roughly the same
- Chunky multiplication can *match* either, but not beat both.

- Must choose to beat either sparse or dense

Cost Analysis

Cost of multiplying f times one chunk of g:

Theorem

Let $f = \sum f_i x^{e_i}$ *and each* $\deg f_i < d_i$.
Let $g \in \mathsf{R}[x]$ *be dense,* $\deg g < m$.

Cost of chunky multiplication $f \cdot g$*:*

$$O\left(m \log \prod_{d_i \le m} (d_i + 1) + (\log m) \sum_{d_i > m} d_i \right)$$

Cost Analysis

Cost of multiplying f times one chunk of g:

Theorem

Let $f = \sum f_i x^{e_i}$ *and each* $\deg f_i < d_i$.
Let $g \in \mathsf{R}[x]$ *be dense,* $\deg g < m$.

Cost of chunky multiplication $f \cdot g$:

$$O\left(m \log \prod_{d_i \le m} (d_i + 1) + (\log m) \sum_{d_i > m} d_i\right)$$

- Minimize $\prod (d_i + 1)$ to compete with dense

Cost Analysis

Cost of multiplying f times one chunk of g:

Theorem

Let $f = \sum f_i x^{e_i}$ *and each* $\deg f_i < d_i$*.*
Let $g \in R[x]$ *be dense,* $\deg g < m$*.*

Cost of chunky multiplication $f \cdot g$*:*

$$O\left(m \log \prod_{d_i \le m} (d_i + 1) + (\log m) \sum_{d_i > m} d_i \right)$$

- Minimize $\prod (d_i + 1)$ to compete with dense
- Minimize $\sum d_i$ to compete with sparse

Converting from Sparse

- $\sum d_i$ minimized in sparse representation
- So introduce slack variable $\omega \geq 1$
- We guarantee $\sum d_i \leq \omega s$.

Comparing Gaps

How to decide if a gap should be collapsed?
Assign "scores" based on

- Maximize *decrease* in $\prod(d_i + 1)$
- Minimize *increase* in $\sum d_i$

Sparse to Chunky Conversion

- Cost $O(s \log s)$ — linear in sparse input size
- Heuristic

Algorithm

1. Split polynomial at every possible gap
2. Assign scores to gaps; put in linked heap
3. While $\sum d_i < \omega s$
4. Collapse gap with best score
5. Update neighboring gaps' scores

Sparse to Chunky Conversion

Example: $f(x) = 5x^3 + 3x^4 - 4x^6 - 8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}$

$$\left[5x^3 + 3x^4\right] \quad \left[-4x^6\right] \quad \left[-8x^{20} + 2x^{21} - 6x^{22}\right] \quad \left[-4x^{24}\right] \quad \left[-5x^{26}\right]$$

Algorithm

1. Split polynomial at every possible gap
2. Assign scores to gaps; put in linked heap
3. While $\sum d_i < \omega s$
4. Collapse gap with best score
5. Update neighboring gaps' scores

Sparse to Chunky Conversion

Example: $f(x) = 5x^3 + 3x^4 - 4x^6 - 8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}$

$$\left[5x^3 + 3x^4\right](36)\left[-4x^6\right](0)\left[-8x^{20} + 2x^{21} - 6x^{22}\right](40)\left[-4x^{24}\right](30)\left[-5x^{26}\right]$$

Algorithm

1. Split polynomial at every possible gap
2. Assign scores to gaps; put in linked heap
3. While $\sum d_i < \omega s$
4. Collapse gap with best score
5. Update neighboring gaps' scores

Sparse to Chunky Conversion

Example: $f(x) = 5x^3 + 3x^4 - 4x^6 - 8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}$

$$\left[5x^3 + 3x^4\right](36)\left[-4x^6\right](0)\left[-8x^{20} + 2x^{21} - 6x^{22} - 4x^{24}\right](30)\left[-5x^{26}\right]$$

Algorithm

1. Split polynomial at every possible gap
2. Assign scores to gaps; put in linked heap
3. While $\sum d_i < \omega s$
4. Collapse gap with best score
5. Update neighboring gaps' scores

Sparse to Chunky Conversion

Example: $f(x) = 5x^3 + 3x^4 - 4x^6 - 8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}$

$$\left[5x^3 + 3x^4\right](36)\left[-4x^6\right](0)\left[-8x^{20} + 2x^{21} - 6x^{22} - 4x^{24}\right](45)\left[-5x^{26}\right]$$

Algorithm

1. Split polynomial at every possible gap
2. Assign scores to gaps; put in linked heap
3. While $\sum d_i < \omega s$
4. Collapse gap with best score
5. Update neighboring gaps' scores

Sparse to Chunky Conversion

Example: $f(x) = 5x^3 + 3x^4 - 4x^6 - 8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}$

$$\left[5x^3 + 3x^4\right](36)\left[-4x^6\right](0)\left[-8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}\right]$$

Algorithm

1. Split polynomial at every possible gap
2. Assign scores to gaps; put in linked heap
3. While $\sum d_i < \omega s$
4. Collapse gap with best score
5. Update neighboring gaps' scores

Sparse to Chunky Conversion

Example: $f(x) = 5x^3 + 3x^4 - 4x^6 - 8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}$

$$\left[5x^3 + 3x^4\right](36)\left[-4x^6\right](0)\left[-8x^{20} + 2x^{21} - 6x^{22} - 4x^{24} - 5x^{26}\right]$$

Algorithm

1. Split polynomial at every possible gap
2. Assign scores to gaps; put in linked heap
3. While $\sum d_i < \omega s$
4. Collapse gap with best score
5. Update neighboring gaps' scores

Converting from Dense

- Finding $\min \prod(d_i + 1)$ non-trivial
- Completely dense rep. has $\prod(d_i + 1) = n + 1$.
- We guarantee $\prod(d_i + 1) < (n + 1)^\omega$
- Idea: Include as many gaps as possible

When to split at a gap?

- Depends heavily on adjacent gaps
- Similar to maze search with backtracking

Dense to Chunky Conversion

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

- Each gap pushed and popped at most once
- At most $n/2$ gaps
- $\therefore$ Complexity $O(n)$ — linear in dense rep. size

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \left[x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \; \left[x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \blacktriangle \left[x^{25} + x^{26} \blacktriangle + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \; \left[x^{25} + x^{26}\right] \blacktriangle \left[x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \left[x^{25} + x^{26}\right] \left[x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \; \left[x^{25} + x^{26}\right] \left[x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \ \left[x^{25} + x^{26}\right] \ \left[x^{29}\right] \blacktriangle \left[x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \; \left[x^{25} + x^{26}\right] \; \left[x^{29}\right] \; \left[x^{31} + x^{32} + x^{33} + x^{34}\right]_{\blacktriangle}$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \; \left[x^{25} + x^{26}\right] \; \left[x^{29}\right] \blacktriangle \left[x^{31} + x^{32} + x^{33} + x^{34}\right] \blacktriangle$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \; \left[x^{25} + x^{26}\right] \left[x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right]_{\blacktriangle} \left[x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]_{\blacktriangle}$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Example: $f = 1 + x + x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}$

$$\left[1 + x\right] \left[x^{25} + x^{26} + x^{29} + x^{31} + x^{32} + x^{33} + x^{34}\right]$$

Algorithm

1. Create empty stack of gaps
2. For each gap in f, moving left to right
3. Pop off all gaps that don't improve $\prod(d_i + 1)$ if polynomial ended here
4. Push current gap onto stack
5. Split at all gaps remaining on stack

Choice of Ring

Assumptions

- Ring elts. have constant storage: $R = \mathbb{Z}_p$
- Ring ops. have unit cost: $p < 2^{30}$
- $\mathsf{M}(n) \in O(n \log n)$: $2^{26} \mid (p-1)$

Implementation Notes

Implemented: Chunky multiplication from dense input using Victor Shoup's NTL

Additions to NTL

- "Lopsided multiplication" to achieve $O(\frac{n}{m}\mathsf{M}(m))$
- Sparse multiplication using heaps (ala Monagan & Pearce)
- In-place multiplication to avoid copying

Conversion Algorithms

1. "Standard" (using "gap stack") with slack var. ω
2. "Naïve" — split at every gap

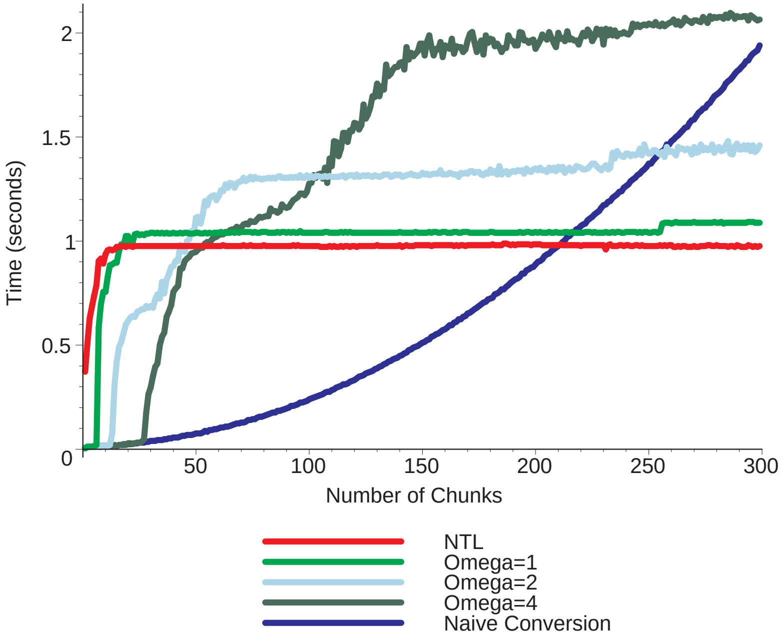
Timing Results

Test Parameters

- Degree fixed at $10\,000$
- 1 to 300 "chunks" in each polynomial
- Degree of each chunk < 10

Algorithms compared:

- Standard NTL Multiplication
- "Standard" chunky with $\omega = 1, 2, 4$
- "Naïve" chunky


Time (seconds)
0
0.5
1
1.5
2
50
100
150
200
250
300
Number of Chunks
NTL
Omega=1
Omega=2
Omega=4
Naive Conversion

Summary

- Adaptive algorithms perform better in easy cases, but never (asymptotically) worse
- Three ideas for adaptive multiplication:
  - Coefficients in sequence
  - Equal-spaced coefficients
  - Chunky coefficients
- Theory does inform practice, to some extent

Future Work

- Compare chunky multiplication to sparse
- Find better gradient between dense/sparse chunky conversion
- Investigate structure of polynomials in practice
- Develop theory further: difficulty measures, relationships
- Combine ideas for adaptive multiplication