Journées Nationales de Calcul Formel — November 14–18, 2011

A unifying algorithm finding all formulae for bilinear computations

Nicolas Estibals
CARAMEL project-team, LORIA, Nancy Université / CNRS / INRIA
Nicolas.Estibals@loria.fr

Joint work with:

Răzvan Bărbulescu
Jérémie Detrey
Paul Zimmermann

Outline of the talk

- Some history
- Formulae for polynomial multiplication
- Enumerating formulae
- Results and conclusion

Some history

- Multiplication is an expensive arithmetic operation
- Well-studied problem
  - Karatsuba (1962)
  - Toom–Cook (1963), evaluation-interpolation schemes
  - CRT-based algorithms
  - Schönhage–Strassen algorithm (1971)
  - ...
- *Five, six-, and seven-term Karatsuba-like formulae*, P. Montgomery (2005)
  - Ad-hoc formulae
  - Exhaustive search for five-term multiplication
  - Non-exhaustive search for six- and seven-term multiplications
  - (January 2011) start a task group to reproduce his search

Outline of the talk

- Some history
- Formulae for polynomial multiplication
- Enumerating formulae
- Results and conclusion

Example: a 2-term polynomial times a 3-term one

- Formula to compute:

$$\begin{aligned} C(X) &= (a_1 \cdot X + a_0) \times (b_2 \cdot X^2 + b_1 \cdot X + b_0) \\ &= a_1 b_2 \cdot X^3 + (a_1 b_1 + a_0 b_2) \cdot X^2 + (a_0 b_1 + a_1 b_0) \cdot X + a_0 b_0 \end{aligned}$$

- 5 products needed only instead of 6
  - Use Karatsuba's trick

$$C(X) = a_1 b_2 \cdot X^3 + (a_1 b_1 + a_0 b_2) \cdot X^2 + ((a_0 + a_1)(b_0 + b_1) - a_1 b_1 - a_0 b_0) \cdot X + a_0 b_0$$

  - Products to compute:

$$\begin{aligned} p_0 &= a_0 \cdot b_0, \\ p_1 &= a_0 \cdot b_2, \\ p_2 &= a_1 \cdot b_1, \\ p_3 &= a_1 \cdot b_2, \\ p_4 &= (a_0 + a_1) \cdot (b_0 + b_1). \end{aligned}$$

  - Reconstructing the result

$$C(X) = p_3 \cdot X^3 + (p_1 + p_2) \cdot X^2 + (p_4 - p_2 - p_0) \cdot X + p_0$$

General form of a multiplication formula

- Formula to compute

$$c_{n+m-2} \cdot X^{n+m-2} + \cdots + c_0 = (a_{n-1} \cdot X^{n-1} + \cdots + a_0) \cdot (b_{m-1} \cdot X^{m-1} + \cdots + b_0)$$

- All formulae for multiplication can be expressed as:
  - Compute some linear combinations of the a_i

$$a'_j = \sum \alpha_{i,j} \cdot a_i$$

  - Compute some linear combinations of the b_i

$$b'_j = \sum \beta_{i,j} \cdot b_i$$

General form of a multiplication formula

- Formula to compute

$$c_{n+m-2} \cdot X^{n+m-2} + \cdots + c_0 = (a_{n-1} \cdot X^{n-1} + \cdots + a_0) \cdot (b_{m-1} \cdot X^{m-1} + \cdots + b_0)$$

- All formulae for multiplication can be expressed as:
  - Compute some linear combinations of the a_i

$$a'_j = \sum \alpha_{i,j} \cdot a_i$$

  - Compute some linear combinations of the b_i

$$b'_j = \sum \beta_{i,j} \cdot b_i$$

  - Perform some products

$$p_j = a'_j \cdot b'_j$$

General form of a multiplication formula

- Formula to compute

$$c_{n+m-2} \cdot X^{n+m-2} + \cdots + c_0 = (a_{n-1} \cdot X^{n-1} + \cdots + a_0) \cdot (b_{m-1} \cdot X^{m-1} + \cdots + b_0)$$

- All formulae for multiplication can be expressed as:
  - Compute some linear combinations of the a_i

$$a'_j = \sum \alpha_{i,j} \cdot a_i$$

  - Compute some linear combinations of the b_i

$$b'_j = \sum \beta_{i,j} \cdot b_i$$

  - Perform some products

$$p_j = a'_j \cdot b'_j$$

  - Reconstruct the result by linearly combining the products

$$c_i = \sum \gamma_{j,i} \cdot p_j$$

General form of a multiplication formula

- Formula to compute

$$c_{n+m-2} \cdot X^{n+m-2} + \cdots + c_0 = (a_{n-1} \cdot X^{n-1} + \cdots + a_0) \cdot (b_{m-1} \cdot X^{m-1} + \cdots + b_0)$$

- All formulae for multiplication can be expressed as:
  - Compute some linear combinations of the a_i

$$a'_j = \sum \alpha_{i,j} \cdot a_i$$

  - Compute some linear combinations of the b_i

$$b'_j = \sum \beta_{i,j} \cdot b_i$$

  - Perform some products

$$p_j = a'_j \cdot b'_j$$

  - Reconstruct the result by linearly combining the products

$$c_i = \sum \gamma_{j,i} \cdot p_j$$

- This is also true for every bilinear application F such that

$$(c_0, \ldots, c_{\ell-1}) = F\left((a_0, \ldots, a_{n-1}), (b_0, \ldots, b_{m-1})\right)$$

Outline of the talk

- Some history
- Formulae for polynomial multiplication
- Enumerating formulae
- Results and conclusion

Formal framework

Formulation in term of vector space for a $n \times m$ multiplication over a given field K

- Represent the coefficients of the result and the products as elements of

  V the nm-dimensional K-vector space generated by $\{a_i b_j\}_{0 \leq i < n, 0 \leq j < m}$

  where the $a_i b_j$'s are seen as formal elements

Formal framework

Formulation in term of vector space for a $n \times m$ multiplication over a given field K

- Represent the coefficients of the result and the products as elements of

  V the nm-dimensional K-vector space generated by $\{a_i b_j\}_{0 \leq i < n, 0 \leq j < m}$

  where the $a_i b_j$'s are seen as formal elements

- Our target: the coefficients of the result is a family

  $$\mathcal{T} = \{c_i\}_{0 \leq i < n+m-1} \subset V$$

  that spans the target subspace $T = \operatorname{Span} \mathcal{T}$ of V

Formal framework

Formulation in term of vector space for a $n \times m$ multiplication over a given field K

- Represent the coefficients of the result and the products as elements of

$$V \text{ the } nm\text{-dimensional } K\text{-vector space generated by } \{a_i b_j\}_{0 \leq i < n, 0 \leq j < m}$$

  where the $a_i b_j$'s are seen as formal elements

- Our target: the coefficients of the result is a family

$$\mathcal{T} = \{c_i\}_{0 \leq i < n+m-1} \subset V$$

  that spans the target subspace $T = \operatorname{Span} \mathcal{T}$ of V

- The set $\mathcal{G}$ of the potential products to use in a formula

$$\mathcal{G}' = \left\{ \left(\sum \alpha_i a_i \right) \cdot \left(\sum \beta_j b_j \right) \;\middle|\; \forall i, \alpha_i \in K \wedge \forall j, \beta_j \in K \right\} \setminus \{0\}$$

  We only consider products modulo a scalar factor

$$\mathcal{G} = \mathcal{G}' / \sim \; \text{ where } p \sim p' \equiv \exists k \in K \text{ s.t. } p = k \cdot p'$$

Formal framework: example

Consider previous example: 2×3 polynomial product in $\mathbb{F}_2[X]$

- V is a 6-dimensional vector space generated by

$$\begin{aligned}\{a_0 \cdot b_0,\quad & a_1 \cdot b_0,\\ a_0 \cdot b_1,\quad & a_1 \cdot b_1,\\ a_0 \cdot b_2,\quad & a_1 \cdot b_2\}\end{aligned}$$

- The target is $\{a_1 b_2, a_1 b_1 + a_0 b_2, a_0 b_1 + a_1 b_0, a_0 b_0\}$

- $\mathcal{G}$ contains 21 products:

$$\begin{aligned}\mathcal{G} = \{ & a_0 \cdot b_0, & & a_1 \cdot b_0, & & (a_1 + a_0) \cdot b_0,\\
& a_0 \cdot b_1, & & a_1 \cdot b_1, & & (a_1 + a_0) \cdot b_1,\\
& a_0 \cdot (b_1 + b_0), & & a_1 \cdot (b_1 + b_0), & & (a_1 + a_0) \cdot (b_1 + b_0),\\
& a_0 \cdot b_2, & & a_1 \cdot b_2, & & (a_1 + a_0) \cdot b_2,\\
& a_0 \cdot (b_2 + b_0), & & a_1 \cdot (b_2 + b_0), & & (a_1 + a_0) \cdot (b_2 + b_0),\\
& a_0 \cdot (b_2 + b_1), & & a_1 \cdot (b_2 + b_1), & & (a_1 + a_0) \cdot (b_2 + b_1),\\
& a_0 \cdot (b_2 + b_1 + b_0), & & a_1 \cdot (b_2 + b_1 + b_0), & & (a_1 + a_0) \cdot (b_2 + b_1 + b_0)\}\end{aligned}$$

Naive algorithm

- Goal: find the optimal formulae (i.e. with a minimum number of products)
  - enumerate the subsets $\mathcal{W} \subset \mathcal{G}$ of exactly k products which give a valid formula
  - for every k until a solution is found

Naive algorithm

- Goal: find the optimal formulae (i.e. with a minimum number of products)
  - enumerate the subsets $\mathcal{W} \subset \mathcal{G}$ of exactly k products which give a valid formula
  - for every k until a solution is found

- Look for $\mathcal{W}$
  - a set of k products

$$\mathcal{W} \subset \mathcal{G} \text{ and } \#\mathcal{W} = k$$

  - that linearly generate the coefficients of the results

$$\mathcal{T} \subset \operatorname{Span} \mathcal{W}$$

Naive algorithm

- Goal: find the optimal formulae (i.e. with a minimum number of products)
  - enumerate the subsets $\mathcal{W} \subset \mathcal{G}$ of exactly k products which give a valid formula
  - for every k until a solution is found
- Look for $\mathcal{W}$
  - a set of k products

$$\mathcal{W} \subset \mathcal{G} \text{ and } \#\mathcal{W} = k$$

  - that linearly generate the coefficients of the results

$$\mathcal{T} \subset \operatorname{Span} \mathcal{W}$$

- Naive approach:
  - enumerate the $\binom{\#\mathcal{G}}{k}$ subsets of cardinal k
  - and test them

Naive algorithm

- Goal: find the optimal formulae (i.e. with a minimum number of products)
  - enumerate the subsets $\mathcal{W} \subset \mathcal{G}$ of exactly k products which give a valid formula
  - for every k until a solution is found

- Look for $\mathcal{W}$
  - a set of k products
$$\mathcal{W} \subset \mathcal{G} \text{ and } \#\mathcal{W} = k$$
  - that linearly generate the coefficients of the results
$$\mathcal{T} \subset \operatorname{Span} \mathcal{W}$$

- Naive approach:
  - enumerate the $\binom{\#\mathcal{G}}{k}$ subsets of cardinal k
  - $\mathcal{G}$ has to be finite
    - ⋆ look for formulae in finite fields K
    - ⋆ take a finite subset of the potential products $\Rightarrow$ May not get all formulae
  - and test them

Naive algorithm

- Goal: find the optimal formulae (i.e. with a minimum number of products)
  - enumerate the subsets $\mathcal{W} \subset \mathcal{G}$ of exactly k products which give a valid formula
  - for every k until a solution is found
- Look for $\mathcal{W}$
  - a set of k products
  $$\mathcal{W} \subset \mathcal{G} \text{ and } \#\mathcal{W} = k$$
  - that linearly generate the coefficients of the results
  $$\mathcal{T} \subset \operatorname{Span} \mathcal{W}$$
- Naive approach:
  - enumerate the $\binom{\#\mathcal{G}}{k}$ subsets of cardinal k
  - $\mathcal{G}$ has to be finite
    - ⋆ look for formulae in finite fields K
    - ⋆ take a finite subset of the potential products ⇒ May not get all formulae
  - and test them
- Drawback
  - Different subsets may span the same subspace

Construct an efficient algorithm: formula spaces

- Look now for subspaces W of V s. t.
  - W can be generated by products: $\text{Span}\,(W \cap \mathcal{G}) = W$
  - only k products are needed: $\dim W = k$
  - contains the target space: $W \supset T$

Construct an efficient algorithm: formula spaces

- Look now for subspaces W of V s. t.
  - W can be generated by products: $\mathrm{Span}\,(W \cap \mathcal{G}) = W$
  - only k products are needed: $\dim W = k$
  - contains the target space: $W \supset T$
- Algorithm

```
procedure extend_to_dim_k(W, H) :
    if dim W = k then
        W is a solution if T ⊂ W
    else
        while H ≠ ∅ :
            Pick a g in H
            if g ∉ W
                extend_to_dim_k(W ⊕ Span(g), H)
end procedure
extend_to_dim_k(∅, G)
```

Construct an efficient algorithm: formula spaces

- Look now for subspaces W of V s. t.
  - W can be generated by products: $\mathrm{Span}\,(W \cap \mathcal{G}) = W$
  - only k products are needed: $\dim W = k$
  - contains the target space: $W \supset T$
- Algorithm

procedure `extend_to_dim_k`(W, $\mathcal{H}$) **:**
    if dim $W = k$ **then**
        W is a solution if $T \subset W$
    else
        while $\mathcal{H} \neq \emptyset$ **:**
            Pick a g in $\mathcal{H}$
            if $g \notin W$
                `extend_to_dim_k`($W \oplus \mathrm{Span}(g)$, $\mathcal{H}$)
end procedure
`extend_to_dim_k`($\emptyset$, $\mathcal{G}$)

- Many formulae could correspond to one solution subspace W
  - each basis of W with elements of $\mathcal{G}$ gives a formula

Construct an efficient algorithm: incomplete basis

- We already know part of W!
  - target space T is a subspace of every solution space W
  - find each W by constructing $\mathcal{I}$ s.t. $W = T \oplus \operatorname{Span} \mathcal{I}$

Construct an efficient algorithm: incomplete basis

- We already know part of W!
  - target space T is a subspace of every solution space W
  - find each W by constructing $\mathcal{I}$ s.t. $W = T \oplus \operatorname{Span} \mathcal{I}$

- Modified algorithm

procedure `extend_to_dim_k`$(W, \mathcal{H})$ **:**
    if $\dim W = k$ **then**
        if $\operatorname{rk}(W \cap \mathcal{G}) = k$ **then**
            W is a solution
    else
        while $\mathcal{H} \neq \emptyset$ **:**
            Pick a g in $\mathcal{H}$
            if $g \notin W$
                `extend_to_dim_k`$(W \oplus \operatorname{Span}(g), \mathcal{H})$
end procedure
`extend_to_dim_k`$(T, \mathcal{G})$

Construct an efficient algorithm: incomplete basis

- We already know part of W!
  - target space T is a subspace of every solution space W
  - find each W by constructing $\mathcal{I}$ s.t. $W = T \oplus \operatorname{Span} \mathcal{I}$
- Modified algorithm

procedure `extend_to_dim_k`$(W, \mathcal{H})$ **:**
  if $\dim W = k$ **then**
    if $\operatorname{rk}(W \cap \mathcal{G}) = k$ **then**
      W is a solution
  else
    while $\mathcal{H} \neq \emptyset$ **:**
      Pick a g in $\mathcal{H}$
      if $g \notin W$
        `extend_to_dim_k`$(W \oplus \operatorname{Span}(g), \mathcal{H})$
end procedure
`extend_to_dim_k`$(T, \mathcal{G})$

- Complexity now depends on

$$\binom{\#\mathcal{G}}{k - \operatorname{rk} \mathcal{T}}$$

Apply our algorithm to 2×3 polynomial multiplication

- Find formulae 2×3 polynomial multiplication in $\mathbb{F}_2[X]$
- Out target: $\mathcal{T} = \{a_1b_2, a_1b_1 + a_0b_2, a_0b_1 + a_1b_0, a_0b_0\}$
  - Rank of the target $\mathcal{T}$ is 4
    - At least, 4 products needed

Apply our algorithm to 2×3 polynomial multiplication

- Find formulae 2×3 polynomial multiplication in $\mathbb{F}_2[X]$
- Out target: $\mathcal{T} = \{a_1 b_2, a_1 b_1 + a_0 b_2, a_0 b_1 + a_1 b_0, a_0 b_0\}$
  - Rank of the target $\mathcal{T}$ is 4
    - ⋆ At least, 4 products needed
  - Attempt with $k = 4$
    - ⋆ $T \cap \mathcal{G} = \{a_0 b_0, a_1 b_2, (a_1 + a_0)(b_2 + b_1 + b_0)\}$
    - ⋆ $\mathrm{rk}\,(T \cap \mathcal{G}) = 3$
    - ⋆ No solutions with $k = 4$ products only

Apply our algorithm to 2×3 polynomial multiplication

- Find formulae 2×3 polynomial multiplication in $\mathbb{F}_2[X]$
- Out target: $\mathcal{T} = \{a_1 b_2, a_1 b_1 + a_0 b_2, a_0 b_1 + a_1 b_0, a_0 b_0\}$
  - Rank of the target $\mathcal{T}$ is 4
    - $\star$ At least, 4 products needed
  - Attempt with $k = 4$
    - $\star$ $T \cap \mathcal{G} = \{a_0 b_0, a_1 b_2, (a_1 + a_0)(b_2 + b_1 + b_0)\}$
    - $\star$ $\text{rk}\,(T \cap \mathcal{G}) = 3$
    - $\star$ No solutions with $k = 4$ products only
  - Attempt with $k = 5$
    - $\star$ Try with $W = T \oplus \text{Span}\,\{a_0 b_1\}$
    - $\star$ $$W \cap \mathcal{G} = \{\begin{array}{lll} a_0 \cdot b_0, & a_1 \cdot b_0, & (a_1 + a_0) \cdot b_0, \\ a_0 \cdot b_1, & a_1 \cdot b_2, & (a_1 + a_0) \cdot (b_2 + b_1), \\ a_0 \cdot (b_1 + b_0), & a_1 \cdot (b_2 + b_0), & (a_1 + a_0) \cdot (b_2 + b_1 + b_0)\} \end{array}$$
    - $\star$ $\text{rk}\,(W \cap \mathcal{G}) = 5$, W is solution!

Apply our algorithm to 2×3 polynomial multiplication

- Find formulae 2×3 polynomial multiplication in $\mathbb{F}_2[X]$
- Out target: $\mathcal{T} = \{a_1 b_2, a_1 b_1 + a_0 b_2, a_0 b_1 + a_1 b_0, a_0 b_0\}$
  - Rank of the target $\mathcal{T}$ is 4
    - At least, 4 products needed
  - Attempt with $k = 4$
    - $T \cap \mathcal{G} = \{a_0 b_0, a_1 b_2, (a_1 + a_0)(b_2 + b_1 + b_0)\}$
    - $\mathrm{rk}\,(T \cap \mathcal{G}) = 3$
    - No solutions with $k = 4$ products only
  - Attempt with $k = 5$
    - Try with $W = T \oplus \mathrm{Span}\,\{a_0 b_1\}$
    - $$W \cap \mathcal{G} = \{ \begin{array}{lll} a_0 \cdot b_0, & a_1 \cdot b_0, & (a_1 + a_0) \cdot b_0, \\ a_0 \cdot b_1, & a_1 \cdot b_2, & (a_1 + a_0) \cdot (b_2 + b_1), \\ a_0 \cdot (b_1 + b_0), & a_1 \cdot (b_2 + b_0), & (a_1 + a_0) \cdot (b_2 + b_1 + b_0)\} \end{array}$$
    - $\mathrm{rk}\,(W \cap \mathcal{G}) = 5$, W is solution!
    - $\{a_0 b_0, a_1 b_0, a_0 b_1, a_1 b_2, (a_1 + a_0)(b_2 + b_1)\}$ form a basis of W which gives a formula
    - There are 3 solution spaces
    - which give a total of 162 formulae

Algorithm works for every bilinear application

- First remark: our algorithm finds all formulae with a given number of products
  - As long as we take all the potential products in $\mathcal{G}$
  - Proves lower bounds on the number of required products

Algorithm works for every bilinear application

- First remark: our algorithm finds all formulae with a given number of products
  - As long as we take all the potential products in $\mathcal{G}$
  - Proves lower bounds on the number of required products

- General algorithm: works for every bilinear application
  - Short products, middle products, cross products
  - Multiplication in complexes, quaternions, field extensions, matrices
  - Multiplication of sparse polynomials and matrices
  - ...

- Also works for applications where the coefficients are quadratic forms
  - Simply requires extending the definition of $\mathcal{G}$

$$\mathcal{G}' = \left\{ \left(\sum \alpha_i a_i\right) \cdot \left(\sum \beta_j a_j\right) \;\middle|\; (\alpha_{n-1}, \ldots, \alpha_0) \preccurlyeq_{\text{lex}} (\beta_{n-1}, \ldots, \beta_0) \right\} \setminus \{0\}$$

  - Apply to the squaring versions of the previous problem
  - Example: squaring of 2-term polynomial

$$\begin{aligned} \mathcal{G} = \{ & a_0 \cdot a_0, \\ & a_0 \cdot a_1, && a_1 \cdot a_1, \\ & a_0 \cdot (a_1 + a_0), && a_1 \cdot (a_1 + a_0), && (a_1 + a_0) \cdot (a_1 + a_0) \} \end{aligned}$$

Real-life example (at least for a crypto Ph.D. student)

- Implementing a pairing over a genus-2 supersingular hyperelliptic curve
- Working in the sextic extension $\mathbb{F}_{2^m}[i, \tau]$
  where $i^2 + i + 1 = 0$ and $\tau^3 + i\tau^2 + i\tau + i = 0$
- Rely on a multiplication algorithm for sparse elements of the form

$$a_3 \cdot \tau^2 + a_2 \cdot \tau + a_1 \cdot i + a_0$$

- Our algorithm exposes an optimal algorithm that necessitates 9 products in $\mathbb{F}_{2^m}$
- Previously known algorithms require at least 11 products

An optimization

Limit the form of the formulae

- Only for symmetric bilinear applications
  - Same number of coefficients in a and b
  - $F\left((a_0, \ldots, a_{n-1}), (b_0, \ldots, b_{n-1})\right) = F\left((b_0, \ldots, b_{n-1}), (a_0, \ldots, a_{n-1})\right)$
  - Verified for multiplication of polynomials of same size
- Only use products with the same linear combination of the a_i's and b_i's

$$\mathcal{G}' = \left\{ \left(\sum \alpha_i a_i\right) \cdot \left(\sum \alpha_i b_i\right) \;\middle|\; \forall i, \alpha_i \in K \right\} \setminus \{0\}$$

  - Reduce the cardinal of $\mathcal{G}$
- Example: 2×2 multiplication in $\mathbb{F}_3[X]$

$$\begin{aligned}\mathcal{G} = \{ & a_0 \cdot b_0, & a_1 \cdot b_0, && (a_1 + a_0) \cdot b_0, && (a_1 - a_0) \cdot b_0, \\ & a_0 \cdot b_1, & a_1 \cdot_1, && (a_1 + a_0) \cdot b_1, && (a_1 - a_0) \cdot b_1, \\ & a_0 \cdot (b_1 + b_0), & a_1 \cdot (b_1 + b_0), && (a_1 + a_0) \cdot b_1 + b_0), && (a_1 - a_0) \cdot (b_1 + b_0), \\ & a_0 \cdot (b_1 - b_0), & a_1 \cdot (b_1 - b_0), && (a_1 + a_0) \cdot (b_1 - b_0), && (a_1 - a_0) \cdot (b_1 - b_0)\}\end{aligned}$$

An optimization

Limit the form of the formulae

- Only for symmetric bilinear applications
  - Same number of coefficients in a and b
  - $F\left((a_0, \ldots, a_{n-1}), (b_0, \ldots, b_{n-1})\right) = F\left((b_0, \ldots, b_{n-1}), (a_0, \ldots, a_{n-1})\right)$
  - Verified for multiplication of polynomials of same size
- Only use products with the same linear combination of the a_i's and b_i's

$$\mathcal{G}' = \left\{ \left(\sum \alpha_i a_i\right) \cdot \left(\sum \alpha_i b_i\right) \;\middle|\; \forall i, \alpha_i \in K \right\} \setminus \{0\}$$

  - Reduce the cardinal of $\mathcal{G}$
- Example: 2×2 multiplication in $\mathbb{F}_3[X]$

$$\begin{aligned}
\mathcal{G} = \{ & a_0 \cdot b_0, & & a_1 \cdot b_0, & & (a_1 + a_0) \cdot b_0, & & (a_1 - a_0) \cdot b_0, \\
& a_0 \cdot b_1, & & a_1{\cdot}_1, & & (a_1 + a_0) \cdot b_1, & & (a_1 - a_0) \cdot b_1, \\
& a_0 \cdot (b_1 + b_0), & & a_1 \cdot (b_1 + b_0), & & (a_1 + a_0) \cdot b_1 + b_0), & & (a_1 - a_0) \cdot (b_1 + b_0), \\
& a_0 \cdot (b_1 - b_0), & & a_1 \cdot (b_1 - b_0), & & (a_1 + a_0) \cdot (b_1 - b_0), & & (a_1 - a_0) \cdot (b_1 - b_0) \}
\end{aligned}$$

Outline of the talk

- Some history
- Formulae for polynomial multiplication
- Enumerating formulae
- Results and conclusion

Computation and results

- Two implementations
  - Generic sage code
  - Core of the algorithm in optimized C with support for multi-threading and large scale distribution

Computation and results

- Two implementations
  - Generic sage code
  - Core of the algorithm in optimized C with support for multi-threading and large scale distribution

- Multiplication over $\mathbb{F}_2[X]$

$\mathbf{n} \times \mathbf{m}$	**Constraints**	$\#\mathcal{G}$	**k**	**# of tests**	**# of subspaces**	**Calculation time [s]**
2×2	None	9	3	1	1	0.00
3×3	None	49	6	9	2	0.00
4×4	None	225	9	$6.60 \cdot 10^3$	4	0.10
5×5	None	961	13	$9.65 \cdot 10^9$	24	$9.90 \cdot 10^5$
	Sym.	31	13	$2.10 \cdot 10^3$	20	0.01
6×6	None	3 969	14	$4.37 \cdot 10^9$	0	$1.85 \cdot 10^6$
	Sym.	63	17	$8.08 \cdot 10^6$	6	54.3
7×7	Sym.	127	22	$3.42 \cdot 10^{12}$	2 460	$5.43 \cdot 10^7$

Conclusion

- General algorithm
- Method that proves lower bounds on the number of subproducts

Conclusion

- General algorithm
- Method that proves lower bounds on the number of subproducts
- Gives all formulae
  - Provides new formulae that cannot be found with previous method
  - We can cherry-pick the one with minimum number of additions

Conclusion

- General algorithm
- Method that proves lower bounds on the number of subproducts
- Gives all formulae
  - Provides new formulae that cannot be found with previous method
  - We can cherry-pick the one with minimum number of additions
- Work in progress and perspectives
  - Lifting formulae for higher-characteristic or characteristic-0 fields
  - Find formulae for your bilinear application!

Thank you for your attention!

Questions?

More results

- Multiplication over $\mathbb{F}_3\,[X]$

$\mathbf{n \times m}$	**Constraints**	$\#\mathcal{G}$	**k**	**# of tests**	**# of subspaces**	**Calculation time [s]**
2×2	None	16	3	1	1	0.00
3×3	None	169	6	24	13	0.00
4×4	None	1 600	9	$4.11 \cdot 10^5$	595	61.9
5×5	None	14 641	11	$4.89 \cdot 10^7$	0	$1.09 \cdot 10^5$
	Sym.	121	12	$3.93 \cdot 10^4$	31	0.71
6×6	Sym.	364	15	$2.37 \cdot 10^8$	3	$1.72 \cdot 10^4$
7×7	Sym.	1 093	16	$1.03 \cdot 10^8$	0	$2.15 \cdot 10^4$

More results

- Multiplication over small extensions of $\mathbb{F}_2$ and $\mathbb{F}_3$
  - Independent of the choice of definition polynomial of the extension

Finite field	Constraints	$\#\mathcal{G}$	k	# of tests	# of subspaces	Calculation time [s]
$\mathbb{F}_{2^2}$	None	9	3	3	3	0.00
$\mathbb{F}_{2^3}$	None	49	6	$7.03 \cdot 10^3$	105	0.02
$\mathbb{F}_{2^4}$	None	225	9	$2.57 \cdot 10^9$	2 025	955
$\mathbb{F}_{2^5}$	None	961	9	$3.10 \cdot 10^{10}$	0	$1.83 \cdot 10^6$
	Sym.	31	13	$3.49 \cdot 10^6$	2 015	13.7
$\mathbb{F}_{2^6}$	Sym.	63	14	$3.78 \cdot 10^9$	0	$2.50 \cdot 10^5$
$\mathbb{F}_{2^7}$	Sym.	127	14	$8.93 \cdot 10^{10}$	0	$1.22 \cdot 10^6$
$\mathbb{F}_{3^2}$	None	16	3	4	4	0.00
$\mathbb{F}_{3^3}$	None	169	6	$2.42 \cdot 10^5$	11 843	5.35
$\mathbb{F}_{3^4}$	None	1 600	7	$6.29 \cdot 10^8$	0	$1.16 \cdot 10^5$
	Sym.	40	9	$1.10 \cdot 10^5$	234	0.98
$\mathbb{F}_{3^5}$	Sym.	121	10	$1.83 \cdot 10^8$	0	$3.77 \cdot 10^3$