What's new in BPAS?

Basic Polynomial Algebra Subprograms

M. Asadi, **A. Brandt**, R.-J. Jing, M. Kazemi, D. Mohajerani,
R.H.C. Moir, M. Moreno Maza, D. Talaashrafi, L. Wang

Ontario Research Center for Computer Algebra
Department of Computer Science
University of Western Ontario, Canada

Friday June 19, 2020

The BPAS Library

BPAS [1] provides **high-performance polynomial algebra**.

→ High performance: core implementation in C, data locality, parallelism

→ Easy to use: **"Dynamic" Object-Oriented interface** in C++ [5]

The BPAS Library

BPAS [1] provides **high-performance polynomial algebra**.

→ High performance: core implementation in C, data locality, parallelism

→ Easy to use: **"Dynamic" Object-Oriented interface** in C++ [5]

Supported operations include:

- FFTs, integer polynomial multiplication, modular dense polynomial arithmetic, real root isolation (old news, see [6])

The BPAS Library

BPAS [1] provides **high-performance polynomial algebra**.

→ High performance: core implementation in C, data locality, parallelism

→ Easy to use: **"Dynamic" Object-Oriented interface** in C++ [5]

Supported operations include:

- FFTs, integer polynomial multiplication, modular dense polynomial arithmetic, real root isolation (old news, see [6])
- Big FFTs: Efficient and parallel FFTs over large characteristic [7]

The BPAS Library

BPAS [1] provides **high-performance polynomial algebra**.

→ High performance: core implementation in C, data locality, parallelism

→ Easy to use: **"Dynamic" Object-Oriented interface** in C++ [5]

Supported operations include:

- FFTs, integer polynomial multiplication, modular dense polynomial arithmetic, real root isolation (old news, see [6])
- Big FFTs: Efficient and parallel FFTs over large characteristic [7]
- **Sparse polynomial arithmetic**, pseudo-division, normal form [3]

The BPAS Library

BPAS [1] provides **high-performance polynomial algebra**.

→ High performance: core implementation in C, data locality, parallelism

→ Easy to use: **"Dynamic" Object-Oriented interface** in C++ [5]

Supported operations include:

- FFTs, integer polynomial multiplication, modular dense polynomial arithmetic, real root isolation (old news, see [6])
- Big FFTs: Efficient and parallel FFTs over large characteristic [7]
- **Sparse polynomial arithmetic**, pseudo-division, normal form [3]
- **Triangular decomposition** via regular chains [2]

The BPAS Library

BPAS [1] provides **high-performance polynomial algebra**.

→ High performance: core implementation in C, data locality, parallelism

→ Easy to use: **"Dynamic" Object-Oriented interface** in C++ [5]

Supported operations include:

- FFTs, integer polynomial multiplication, modular dense polynomial arithmetic, real root isolation (old news, see [6])
- Big FFTs: Efficient and parallel FFTs over large characteristic [7]
- **Sparse polynomial arithmetic**, pseudo-division, normal form [3]
- **Triangular decomposition** via regular chains [2]
- **Lazy multivariate power series**, Weierstrass preparation, factorization via Hensel's lemma [4]

The BPAS Library

BPAS [1] provides **high-performance polynomial algebra**.

→ High performance: core implementation in C, data locality, parallelism

→ Easy to use: **"Dynamic" Object-Oriented interface** in C++ [5]

Supported operations include:

- FFTs, integer polynomial multiplication, modular dense polynomial arithmetic, real root isolation (old news, see [6])
- Big FFTs: Efficient and parallel FFTs over large characteristic [7]
- **Sparse polynomial arithmetic**, pseudo-division, normal form [3]
- **Triangular decomposition** via regular chains [2]
- **Lazy multivariate power series**, Weierstrass preparation, factorization via Hensel's lemma [4]
- Fourier-Motzkin elimination: new algorithm, complexity measures [8]

C++ Templates for a Dynamic, Friendly Interface [5]

Motivation: encode the algebraic hierarchy as a class hierarchy for ease-of-use

$$\text{field} \subset \text{Euclidean domain} \subset \text{GCD domain} \subset \text{integral domain} \subset \text{ring}$$

C++ Templates for a Dynamic, Friendly Interface [5]

Motivation: encode the algebraic hierarchy as a class hierarchy for ease-of-use

$$\text{field} \subset \text{Euclidean domain} \subset \text{GCD domain} \subset \text{integral domain} \subset \text{ring}$$

Goal: maintain strict object type safety and *mathematical type safety*.

→ via polymorphism, a simple hierarchy has no mathematical type safety

→ e.g. $\mathbb{Z}/17\mathbb{Z}$, $\mathbb{Q}[x]$ are both Euclidean domains, but incompatible.

C++ Templates for a Dynamic, Friendly Interface [5]

Motivation: encode the algebraic hierarchy as a class hierarchy for ease-of-use

field ⊂ Euclidean domain ⊂ GCD domain ⊂ integral domain ⊂ ring

Goal: maintain strict object type safety and *mathematical type safety*.

→ via polymorphism, a simple hierarchy has no mathematical type safety

→ e.g. $\mathbb{Z}/17\mathbb{Z}$, $\mathbb{Q}[x]$ are both Euclidean domains, but incompatible.

Solution: *curiously recurring template pattern*, template metaprogramming

→ Abstract class hierarchy encodes the interface of each algebraic type

→ Template parameter makes interface mathematically safe at compile-time

→ Polys automatically decide algebraic type (superclass) based on ground ring

```
template <class Derived>
class BPASRing {    Derived add(Derived x, Derived y)    };

template <class Derived>
class BPASEuclideanDomain : BPASGCDDomain<Derived>;

class Integer : BPASEuclideanDomain<Integer>;

template <class Ring, Deriverd>
class Polynomial : conditional< is_base_of<Ring, BPASField<Ring>,
     BPASEuclideanDomain<Dervied>, BPASRing<Derived> >;
```

Sparse Multivariate Polynomial Arithmetic [3]

Goal: highly efficient operations on multivariate polynomials to underlie polynomial system solving, regular chains, triangular decomposition

Sparse Multivariate Polynomial Arithmetic [3]

Goal: highly efficient operations on multivariate polynomials to underlie polynomial system solving, regular chains, triangular decomposition

Solution: "Alternating Array" for $\mathbb{Q}[\underline{X}]$, $\mathbb{Z}[\underline{X}]$ using exponent packing

- $\rightarrow$ per term data locality (cf. separate locality for coefficients and monomials)
- $\rightarrow$ optimizes Monagan's heap arithmetic [9]; extends to pseudo-division & n.f.
- $\rightarrow$ *dynamic unpacking* (unpublished) supports *unlimited* variables (cf. 31 in [9])

$$13x^2y^3z^2 + 5x^2y + 7y^3z + 11yz^4 := \underbrace{\boxed{13}\boxed{2|3|2}}_{\text{Term 1}}\underbrace{\boxed{5}\boxed{2|1|0}}_{\text{Term 2}}\underbrace{\boxed{7}\boxed{0|3|1}}_{\text{Term 3}}\underbrace{\boxed{11}\boxed{0|1|4}}_{\text{Term 4}}$$

Sparse Multivariate Polynomial Arithmetic [3]

Goal: highly efficient operations on multivariate polynomials to underlie polynomial system solving, regular chains, triangular decomposition

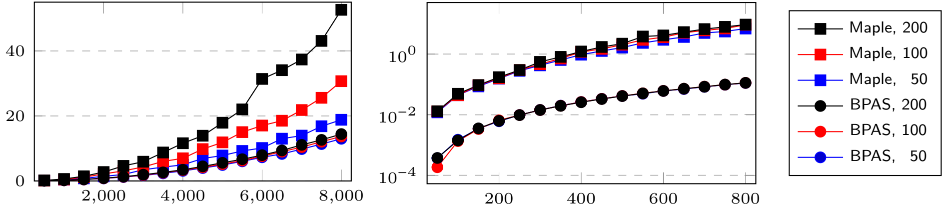
Solution: "Alternating Array" for $\mathbb{Q}[\underline{X}]$, $\mathbb{Z}[\underline{X}]$ using exponent packing

- $\rightarrow$ per term data locality (cf. separate locality for coefficients and monomials)
- $\rightarrow$ optimizes Monagan's heap arithmetic [9]; extends to pseudo-division & n.f.
- $\rightarrow$ *dynamic unpacking* (unpublished) supports *unlimited* variables (cf. 31 in [9])

$$13x^2y^3z^2 + 5x^2y + 7y^3z + 11yz^4 :=$$

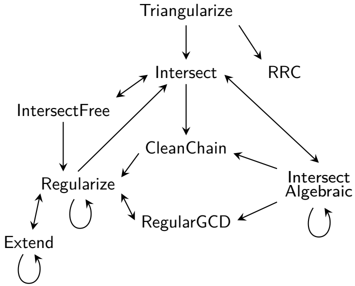
13	2|3|2	5	2|1|0	7	0|3|1	11	0|1|4
Term 1		Term 2		Term 3		Term 4	

$\mathbb{Z}[X_1, \ldots, X_5]$ multiplication and Euclidean division (time (s) vs number of terms; varying sparsity)



Parallel Triangular Decompositions [2]

Goal: parallelism in triangular decomposition despite geometric problems



Parallel Triangular Decompositions [2]

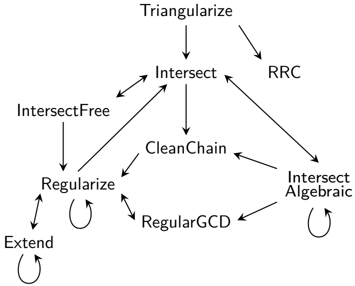
Goal: parallelism in triangular decomposition despite geometric problems

Solution: exploit as much as is available

→ Coarse-grained: parallel calls intersects on independent components.

→ Fine-grained: *asynchronous generators* form a pipeline among subroutines.

→ DnC: removal of redundant components



Parallel Triangular Decompositions [2]

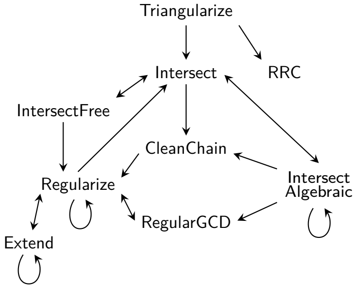
Goal: parallelism in triangular decomposition despite geometric problems

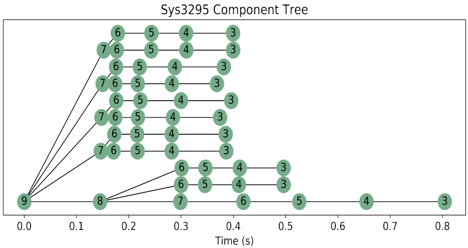
Solution: exploit as much as is available

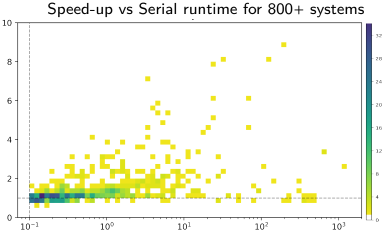
→ Coarse-grained: parallel calls intersects on independent components.

→ Fine-grained: *asynchronous generators* form a pipeline among subroutines.

→ DnC: removal of redundant components







Lazy Multivariate Power Series [4]

Goal: efficient multivariate power series and univar. polynomials over PS

Lazy Multivariate Power Series [4]

Goal: efficient multivariate power series and univar. polynomials over PS

Solution: a lazy design in C (compiled vs. scripting like MAPLE)

Lazy Multivariate Power Series [4]

Goal: efficient multivariate power series and univar. polynomials over PS

Solution: a lazy design in C (compiled vs. scripting like MAPLE)

→ PS stored as dense array of homogeneous parts and a function pointer

→ PS hold "ancestors" to call ancestor generator in its own generator

Lazy Multivariate Power Series [4]

Goal: efficient multivariate power series and univar. polynomials over PS

Solution: a lazy design in C (compiled vs. scripting like MAPLE)

→ PS stored as dense array of homogeneous parts and a function pointer

→ PS hold "ancestors" to call ancestor generator in its own generator

→ Supports lazy Weierstrass preparation and lazy factorization via Hensel's lemma. Factors are returned immediately and can be updated dynamically, without any re-computation.

Lazy Multivariate Power Series [4]

Goal: efficient multivariate power series and univar. polynomials over PS

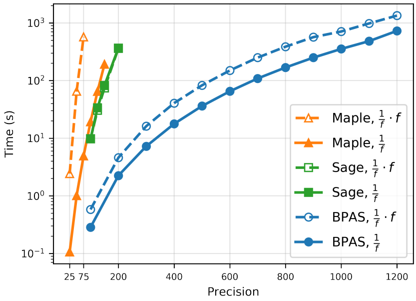
Solution: a lazy design in C (compiled vs. scripting like MAPLE)

→ PS stored as dense array of homogeneous parts and a function pointer

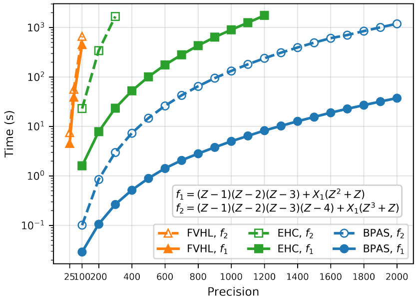
→ PS hold "ancestors" to call ancestor generator in its own generator

→ Supports lazy Weierstrass preparation and lazy factorization via Hensel's lemma. Factors are returned immediately and can be updated dynamically, without any re-computation.

Power Series Arithmetic for $f = 1 + X_1 + X_2 + X_3$



Factorization via Hensel's Lemma in $\mathbb{Q}[[X_1]][Z]$



References

[1] M. Asadi, A. Brandt, C. Chen, S. Covanov, M. Kazemi, F. Mansouri, D. Mohajerani, R. H. C. Moir, M. Moreno Maza, D. Talaashrafi, L. Wang, N. Xie, and Y. Xie. *Basic Polynomial Algebra Subprograms (BPAS)*. `http://www.bpaslib.org`. 2020.

[2] M. Asadi, A. Brandt, R. H. C. Moir, M. Moreno Maza, and Y. X. "On the Parallelization of Triangular Decomposition of Polynomial Systems". In: *ISSAC 2020, Proceedings*. (to appear). 2020.

[3] M. Asadi, A. Brandt, R. H. C. Moir, and M. Moreno Maza. "Algorithms and Data Structures for Sparse Polynomial Arithmetic". In: *Mathematics* 7.5 (2019), p. 441.

[4] A. Brandt, M. Kazemi, and M. Moreno Maza. "Power Series Arithmetic with the BPAS Library". In: *CASC 2020*. (submitted). 2020.

[5] A. Brandt, R. H. C. Moir, and M. Moreno Maza. "Employing C++ Templates in the Design of a Computer Algebra Library". In: *Mathematical Software – ICMS 2020*. (to appear). 2020.

[6] C. Chen, S. Covanov, F. Mansouri, M. Moreno Maza, N. Xie, and Y. Xie. "The Basic Polynomial Algebra Subprograms". In: *ICMS 2014, Proceedings*. 2014, pp. 669–676.

[7] S. Covanov, D. Mohajerani, M. Moreno Maza, and L. Wang. "Big Prime Field FFT on Multi-core Processors". In: *ISSAC 2019, Proceedings*. 2019, pp. 106–113.

[8] R.-J. Jing, M. Moreno Maza, and D. Talaashrafi. "Complexity Estimates for Fourier-Motzkin Elimination". In: *CASC 2020*. (submitted). 2020.

[9] M. B. Monagan and R. Pearce. "Sparse polynomial division using a heap". In: *J. Symb. Comput.* 46.7 (2011), pp. 807–822.