Sparse Polynomial Arithmetic with the BPAS Library

Mohammadali Asadi, Alexander Brandt,
Robert H.C. Moir, Marc Moreno Maza

Ontario Research Center for Computer Algebra
University of Western University, London, Ontario

IBM Center for Advanced Studies, Markham, Ontario

CASC 2018, Université de Lille, September 19, 2018

The BPAS Library

Basic Polynomial Algebra Subprograms

The Basic Polynomial Algebra Subprograms (BPAS) is a free, open-source library providing support for polynomial arithmetic and system solving.

- ↳ Optimized fundamental operations support higher-level algorithms.
- ↳ Dense and **sparse** polynomial multiplication, division, pseudo-division.
- ↳ Normal forms, subresultants, regular chains.

`http://www.bpaslib.org`

Outline

Motivation

- To support the developement of our polynomial system solver we look to optimize some fundamental operations.
- Build from the ground up:
  - ↳ Polynomial multiplication, division, pseudo-division;
  - ↳ Normal forms, pseudo-division w.r.t a triangular set;
  - ↳ Subresultants (in progress);
  - ↳ Regular chains.
- For multivariate polynomials, sparsity must be exploited when possible.

Motivation

- Algorithm performance on modern computers is influenced by the *processor-memory gap* [2] and by the *memory wall* [8].
- Amount of memory used and how that memory is traversed is very important.
  ↳ Data locality, cache misses.
- Sparse algorithms, like that of Johnson [4], naturally make use of locality by producing the terms of the sum (difference, product, quotient, remainder) *in-order*.

Related and Previous Works

- Stephen C Johnson. "Sparse polynomial arithmetic". In: *ACM SIGSAM Bulletin* 8.3 (1974), pp. 63–71
- Michael Monagan and Roman Pearce. "Sparse polynomial division using a heap". In: *J. Symb. Comput.* 46.7 (2011), pp. 807–822
- Michael Monagan and Roman Pearce. "Polynomial division using dynamic arrays, heaps, and packed exponent vectors". In: *CASC 2007*. Springer. 2007, pp. 295–315
- Michael Monagan and Roman Pearce. "The design of Maple's sum-of-products and POLY data structures for representing mathematical objects". In: *ACM Communications in Computer Algebra* 48.3/4 (2015), pp. 166–186

From this work, MAPLE has become a leader in arithmetic performance and we use it as base of comparison.

Notations

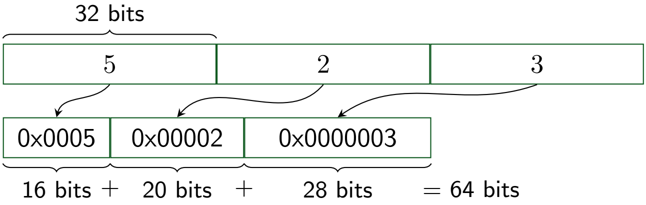
Throughout this presentation we use the following notation:

$$a = \sum_{i=1}^{n} a_i X^{\alpha_i} = \sum_{i=1}^{n} A_i$$

- The polynomial a has n terms.
- a_i are non-zero coefficients.
- α_i are **exponent vectors** for the variables $X = (x_1, x_2, \dots, x_m)$
- A_i are non-zero terms.
- $lm(a) = X^{\alpha_1}$ is the *leading monomial* of a.
- $lc(a) = a_1$ is the *leading coefficient* of a.
- $lt(a) = a_1 X^{\alpha_1} = A_1$ is the *leading term* of a.

Polynomial Data Structures: The Basics

- Polynomials are essentially collections of individual terms.
  - ↳ Must encode individual terms efficiently,
  - ↳ Must collect terms into a polynomial data structure effectively.
- An individual term is simply a coefficient and monomial.
  - ↳ Coefficients easily represented by GMP numbers.
  - ↳ With a consistent variable ordering, only exponent vectors need be encoded. This is done using exponent packing [1, 3] — bit-tricks.

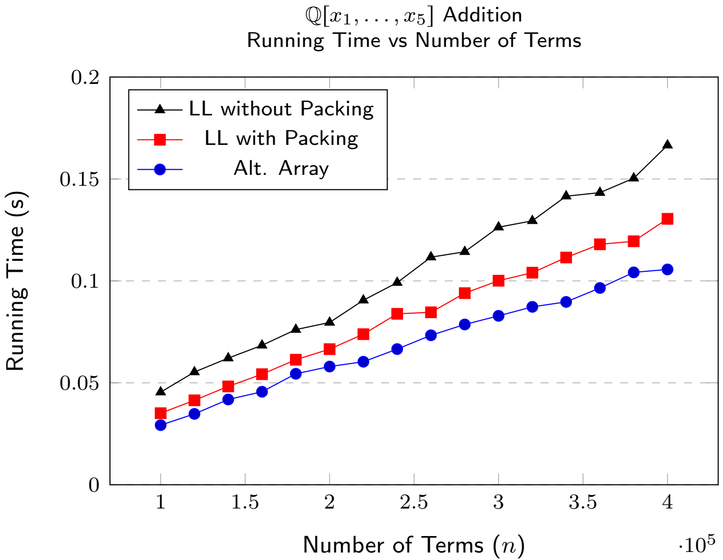


Polynomial Data Structures: The Collection

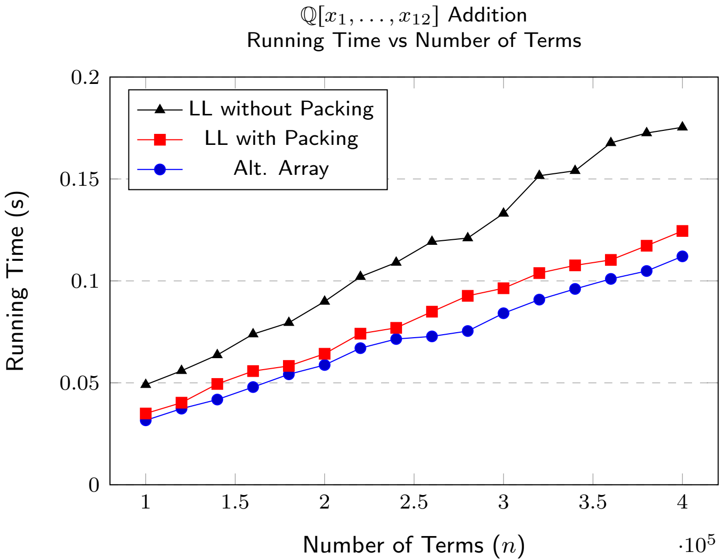
- A naïve approach would be to collect each term into a linked list.
  - ↳ Wasted memory storing a pointer for each node.
  - ↳ Indirection of pointers means adjacent terms not necessarily adjacet in memory.
- An *alternating array* is more succinct, removing pointers and alternating between coefficients and exponent vectors side-by-side in an array.
  - ↳ Adjacent terms are adjacent in memory, no overhead in encoding the structure (no pointers).
- In either case, maintain a canonical representation by keeping terms sorted using some term-order (lex).

$$\underbrace{\boxed{13}\boxed{x^2y^3}}_{\text{Term 1}}\underbrace{\boxed{5}\boxed{x^2y}}_{\text{Term 2}}\underbrace{\boxed{7}\boxed{y^3z}}_{\text{Term 3}}\boxed{\quad\cdots\quad}$$

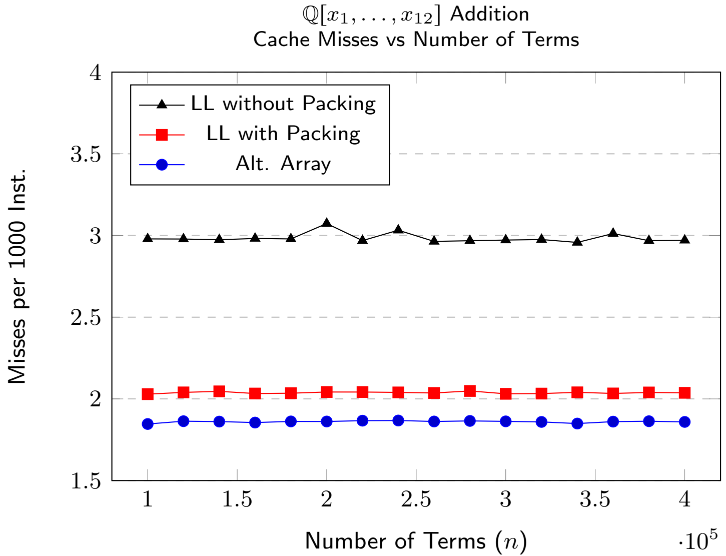
Polynomial Data Structures: Experimentation



Polynomial Data Structures: Experimentation



Polynomial Data Structures: Experimentation

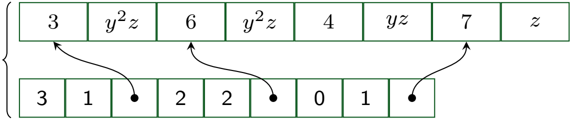


Polynomial Data Structures: Recursive Views

- Many polynomial operations are essentially univariate, viewing multivaraite polynomials recursively: pseudo-division, subresultants.

- It is useful to view a multivariate polynomial recursively, creating a univariate polynomial with polynomial coefficients.
  - ↳ Reuse the original polynomial array as coefficients → fast conversion
  - ↳ Create a small auxiliary alternating array for univariate exponents, coefficient size, and pointers to the original array.

Recursive View



$3x^3y^2z + 6x^2y^2z + 4x^2yz + 7z$ encoded recursively in x.

Sparse Polynomial Addition

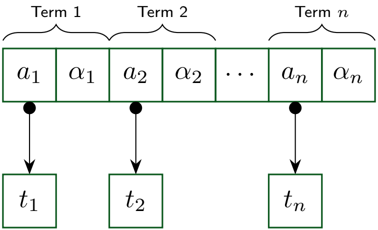
- Sparse polynomial addition highlights the general idea of our sparse arithmetic scheme: *generate terms in order* to exploit locality.

- Addition is essentially one step of merge-sort.
  - If terms are equal they are combined and appended to sum, otherwise simply append the maximum of the two.

$$\left.\begin{array}{l} a = 13x^2y^3 + 5x^2y \quad + \underset{\uparrow}{7}y^3z + \dots \\[2em] b = 6x^2y \quad\quad + \underset{\uparrow}{1}2xz^2 + \;\; 4y^2 \;\; + \dots \end{array}\right\} \; c = 13x^2y^3 + 11x^2y + \underset{\uparrow}{}$$

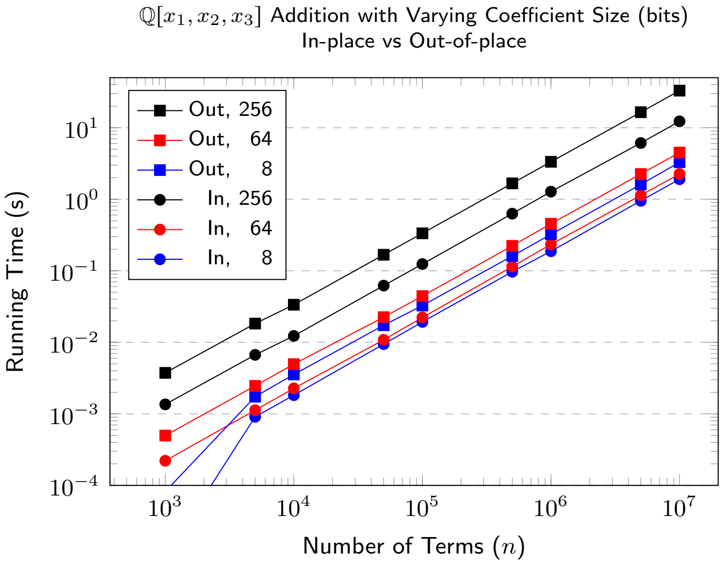
Polynomial Data Structures: "In-Place" Arithmetic

- In reality, GMP numbers are encoded as a *head* and a *tree*.
  - ↳ The head contains metadata: size, allocation, pointer to actual data.
  - ↳ The tree contains the actual encoding.



- For a lazy in-place implementation, simply copy all meta data *including pointers to trees*, do not copy the trees.

Sparse Addition: Experimentation



Sparse Polynomial Multiplication

- Multiplication performs an n-way merge of "streams", each stream being one term of a distributed over the entirety of b.
- Since b is sorted, multiplying by a single term of a keeps the sort order.
- Instead of choosing the maximum of 2 partitions as in addition, we choose the maximum among n.
  - ↳ This choice can be effectively implemented using a **heap**.
  - ↳ Optimize the heap $\implies$ Optimize arithmetic performance.

$$a \cdot b = \begin{cases} (a_1 \cdot b_1)\, X^{\alpha_1+\beta_1} + (a_1 \cdot b_2)\, X^{\alpha_1+\beta_2} + (a_1 \cdot b_3)\, X^{\alpha_1+\beta_3} + \dots \\ (a_2 \cdot b_1)\, X^{\alpha_2+\beta_1} + (a_2 \cdot b_2)\, X^{\alpha_2+\beta_2} + (a_2 \cdot b_3)\, X^{\alpha_2+\beta_3} + \dots \\ \quad \vdots \\ (a_n \cdot b_1)\, X^{\alpha_n+\beta_1} + (a_n \cdot b_2)\, X^{\alpha_n+\beta_2} + (a_n \cdot b_3)\, X^{\alpha_n+\beta_3} + \dots \end{cases}$$

Sparse Polynomial Arithmetic: Quantifying Sparsity

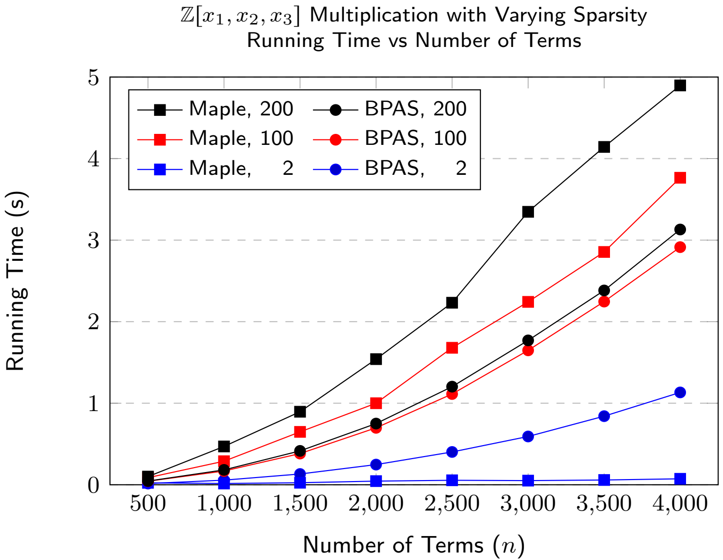
- For univariate polynomials, its sparsity can easily be described using the degree difference between any two adjacent terms. We quantify sparisty as the smallest integer larger than any such difference.

$$x^{12} + x^9 + x^2 + 1 \qquad \left.\begin{aligned} 12 - 9 &= 3 \\ \mathbf{9} - \mathbf{2} &= \mathbf{7} \\ 2 - 0 &= 3 \end{aligned}\right\} \implies \text{sparsity } = 8$$

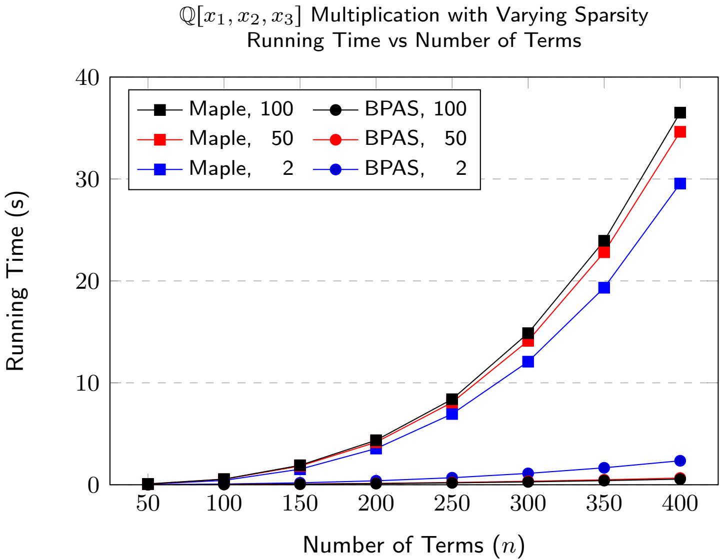
- *Kronecker Substituion* can map a multivariate polynomial to a univariate one in order to apply the same notion of sparsity. With d as an upper bound on partial degrees:

$$x_1^{e_1} x_2^{e_2} \ldots x_m^{e_m} \rightarrow y^{e_1 \cdot d \,+\, e_2 \cdot d^2 \,+\, \ldots \,+\, e_m \cdot d^m}$$

Sparse Polynomial Multiplication: Experimentation



Sparse Polynomial Multiplication: Experimentation



Sparse Polynomial Division

$$a = qb + r,\ deg(r) < deg(b)$$

Division is simply an application of multiplication.

- The quotient begins as $q^{(1)} = \mathrm{lt}(a)/\mathrm{lt}(b)$.
- Quotient and remainder terms are then produced from $q \cdot b$ with q updating throughout the computation.

The main idea is to repeatedly choose the leading term of $a - q^{(i)}b$

- Each new term of the quotient, q_{i+1}, is constructed to exactly cancel the leading term of $a - q^{(i)}b$

$$\begin{array}{r@{\,}l}
 & \underline{\ 3x^2 + 1\qquad\ } = q \\
2x & \big)\, 6x^3 + 2x + 1 \\
 & \underline{-3x^2(2x)\qquad\quad} \\
 & \qquad\qquad 2x + 1 \\
 & \underline{\qquad\quad -1(2x)\ \ } \\
 & \qquad\qquad\quad\ 1 = r
\end{array}$$

Sparse Polynomial Division

$$a = bq + r$$

Naïve Division

$q := 0$; $r := 0$
while $(\tilde{r} := \mathrm{lt}(a - qb - r)) \neq 0$ **do**
  if $\mathrm{lt}(b) \mid \tilde{r}$ **then**
    $q := q + \tilde{r}/\mathrm{lt}(b)$
  else
    $r := r + \tilde{r}$
end
return (q, r)

Heap Division

$(q, r, l) := 0$
$k := 1$
while $(\delta := \textbf{heapPeek}()) > -1$ **or** $k \leqslant n_a$ **do**
  if $\delta < \alpha_k$ **then**
    $\tilde{r} := A_k$
    $k := k + 1$
  else if $\delta = \alpha_k$ **then**
    $\tilde{r} := A_k - \textbf{heapExtract}()$
    $k := k + 1$
  else
    $\tilde{r} := -\textbf{heapExtract}()$
  if $B_1 \mid \tilde{r}$ **then**
    $\ell := \ell + 1$
    $Q_\ell := \tilde{r}/B_1$
    $q := q + Q_\ell$
    $\textbf{heapInsert}(Q_\ell, B_2)$
  else
    $r := r + \tilde{r}$
end
return (q, r)

Sparse Pseudo-Division

$$h^{deg(a)-deg(b)+1}a = qb + r,\ deg(r) < deg(b)$$

- The division algorithms can easily be adapted to pseudo-division. Repeatedly choose the leading term of $h^i a - q^{(i)}b,\ h = lc(b)$.
- However, performance is not as easily attained.
  - ↳ Delay multiplication by h as long as possible, performing only when strictly necessary.
  - ↳ Uses recursive data structure for efficient recursive, univariate view.
  - ↳ Careful implementation of *in-place* arithmetic to minimize memory movement when updating quotient and remainder.
- A *sparse* pseudo-division computes h^ℓ where $\ell \leqslant deg(a) - deg(b) + 1$ is the actual number of division steps performed.

Sparse Polynomial Arithmetic: Pseudo-Division

$$h^\ell a = qb + r$$

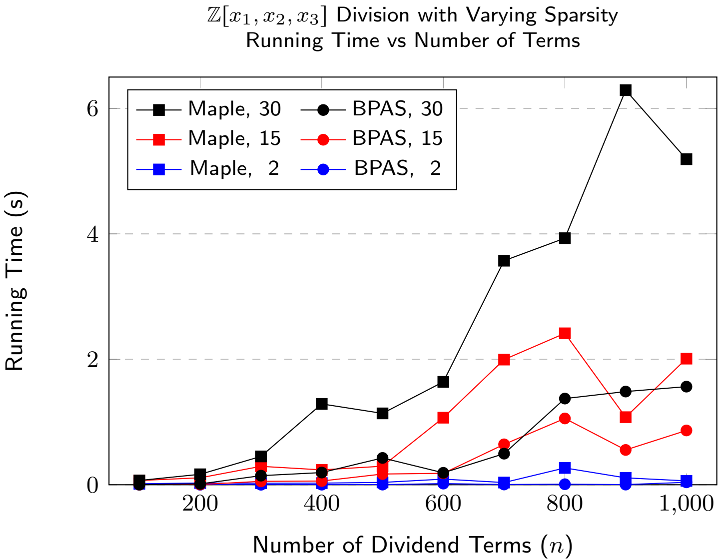
Naïve Pseudo-Division

$(q, r, \ell) := 0$
$h := \mathrm{lc}(b)$; $\gamma = \deg(b)$
while $(\tilde{r} := \mathrm{lt}(h^\ell a - qb - r)) \neq 0$ **do**
  if $x^\gamma \mid \tilde{r}$ **then**
    $q := hq + \tilde{r}/x^\gamma$
    $\ell := \ell + 1$
  else
    $r := r + \tilde{r}$
end
return (q, r, ℓ)

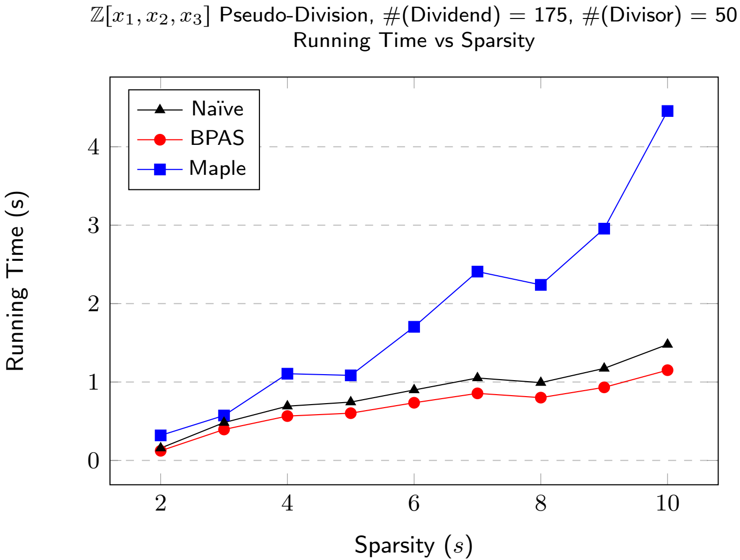
Heap Pseudo-Division

$(q, r, l) := 0$; $k := 1$
$h := \mathrm{lc}(b)$; $\gamma := \deg(b)$
while $(\delta := \mathbf{heapPeek}()) > -1$ **or** $k \leqslant n_a$ **do**
  if $\delta < \alpha_k$ **then**
    $\tilde{r} := h^\ell A_k$
    $k := k + 1$
  else if $\delta = \alpha_k$ **then**
    $\tilde{r} := h^\ell A_k - \mathbf{heapExtract}()$
    $k := k + 1$
  else
    $\tilde{r} := -\mathbf{heapExtract}()$
  if $x^\gamma \mid \tilde{r}$ **then**
    $q := hq$
    $\ell := \ell + 1$
    $Q_\ell := \tilde{r}/x^\gamma$
    $q := q + Q_\ell$
    $\mathbf{heapInsert}(Q_\ell, B_2)$
  else
    $r := r + \tilde{r}$
end
return (q, r, ℓ)

Sparse Polynomial Division: Experimentation



Sparse Pseudo-Division: Experimentation



Thank you!

Questions?

References

[1] Andrew D Hall Jr. "The ALTRAN system for rational function manipulation-a survey". In: *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*. ACM. 1971, pp. 153–157.

[2] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. 4th. Elsevier, 2017.

[3] Joris van der Hoeven and Grégoire Lecerf. "On the bit-complexity of sparse polynomial and series multiplication". In: *J. Symb. Comput.* 50 (2013), pp. 227–254. DOI: 10.1016/j.jsc.2012.06.004. URL: https://doi.org/10.1016/j.jsc.2012.06.004.

[4] Stephen C Johnson. "Sparse polynomial arithmetic". In: *ACM SIGSAM Bulletin* 8.3 (1974), pp. 63–71.

[5] Michael Monagan and Roman Pearce. "Polynomial division using dynamic arrays, heaps, and packed exponent vectors". In: *CASC 2007*. Springer. 2007, pp. 295–315.

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

[7] Michael Monagan and Roman Pearce. "The design of Maple's sum-of-products and POLY data structures for representing mathematical objects". In: *ACM Communications in Computer Algebra* 48.3/4 (2015), pp. 166–186.

[8] Wm A Wulf and Sally A McKee. "Hitting the memory wall: implications of the obvious". In: *ACM SIGARCH computer architecture news* 23.1 (1995), pp. 20–24.