FORMapart – a FORM package and the lessons we learnt writing it

Adam Kardos

in collaboration with

Levente Fekésházy

arXiv:2606.xxxxx (to appear)

FORM Developers' Workshop 2026
23 June 2026

UNIVERSITY of DEBRECEN

Contents

- Motivation: partial fractioning without spurious singularities
- The FORM implementation: design and tricks
- Testing and instrumentation
- Benchmarks
- Usage, summary and outlook

This is a FORM-centric talk: expect term streams, per-term dollars, ctables, `tform` and a fair number of code snippets.

UNIVERSITY of DEBRECEN

Motivation & algorithm

Why partial fractioning?

- Multi-loop amplitudes $\rightarrow$ IBP reduction $\rightarrow$ huge coefficient matrices of rational functions in many variables (ε, invariants, masses)
- These coefficients are dominated by their denominators; a good partial-fraction form
  - drastically shrinks the expression ($\sim 25\times$ reported on the benchmark below)
  - exposes the pole structure for subsequent integration / ε-expansion
- Multivariate denominators are linear in each integration variable in a wide and important class of problems (dlog bases, Feynman parametrization, $\mathcal{U}/\mathcal{F}$ polynomials)

$\Rightarrow$ Goal: a fast, exact, multivariate partial-fraction tool for linear denominators

UNIVERSITY of DEBRECEN

The problem with naive partial fractioning

- Naive iterated single-variable apart invents denominators absent from the original. Decompose $\dfrac{1}{(x+y)(x-y)}$ in x, y:

$$\frac{1}{(x+y)(x-y)} \xrightarrow{\text{apart in } x} \frac{1}{2y\,(x-y)} - \frac{1}{2y\,(x+y)}$$

- The new $\dfrac{1}{y}$ is spurious: y is not an original denominator, and $y = 0$ is not a singularity of the original
- Spurious poles cancel only in the full sum; they wreck term-by-term numerics and inflate the size, and the result is variable-order dependent

$\Rightarrow$ `LinApart3` keeps $\{x+y,\ x-y\}$ as a valid basis: only original denominators appear, at most n per term (n = active variables)

UNIVERSITY *of* DEBRECEN

The algorithm in a nutshell

Write each $D_i = c_i + \sum_k C_{ik}\, x_k$ (linear in the active variables).

- **Phase 1: null relation elimination.** The coefficient matrix C has a (left) null space. A null vector cancels the variable part, leaving $\sum_i \lambda_i D_i = \sum_i \lambda_i c_i \equiv \text{const}$; this lets us remove one denominator from a product, recursively, until $\leq$ n distinct denominators remain.
- **Phase 2: basis identification & residues.** For each surviving n-denominator basis, change to the basis (a matrix inverse / adjugate) and extract the residues.
- **Numerators, deferred.** Variable-dependent numerators are absorbed (expanded in the basis coordinates) in a separate step after the residues

– `LinApart`, `LinApart2`: univariate; `LinApart3`: multivariate (this work)

→ `LinApart3` ships a Mathematica and a native `FORM` implementation (this talk)

UNIVERSITY *of* DEBRECEN

Why FORM?

- The coefficients that need apart are produced by FORM pipelines, staying in FORM avoids costly hand-off and re-parsing
- FORM is built for very large expressions that do not fit in RAM (term streaming, disk sort)
- Scales on multicore via `tform`; open source, public domain, active development (FORM 5)
- But FORM is not Mathematica: no global expression tree, no random-access mutation. The whole design has to be rethought as a term-rewriting stream

$\Rightarrow$ The rest of the talk: how the LinApart algorithm maps onto FORM, and the tricks that make it work

UNIVERSITY of DEBRECEN

The FORM implementation

UNIVERSITY of DEBRECEN

Design: a term stream, not an expression tree

Mathematica

- one global expression tree
- random-access lookup & mutation
- recurse / replace anywhere

`FORM`

- expression = stream of terms
- a module rewrites the terms; its `.sort` then cancels & combines them and the stream restarts
- terms processed in parallel, independently

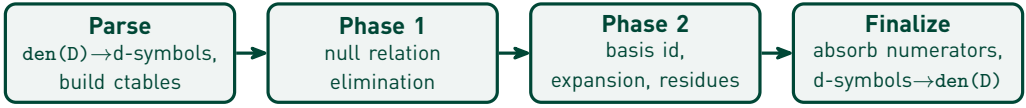
The implementation is built around one split:

- Fixed data (built once) lives in global `ctables`: coefficient matrix, denominator table, elimination order, null-relation cache
- Per-term bookkeeping lives in per-term local `$`-variables, seeded from the term's own structure as it streams by

⇒ ctables: read-mostly. dollars: per-term, per-worker.

UNIVERSITY of DEBRECEN

The pipeline



- One driver procedure, `ApartMultiLinApart(den, x, y,…)`, orchestrates the four stages
- It acts on every visible local expression, a term-stream operator, not a function call on one object

⇒ ∼50 small single-purpose `#procedures` underneath, each composable and separately tested (we time the stage boundaries later)

UNIVERSITY of DEBRECEN

Trick #1: the d-symbol representation

```
* user cfunction, multiplicity = integer exponent
Local F = den(x)^2*den(y)*den(x+y)*den(x+y-1);
* parser: den(D_i) -> APd_i,  APd_i = 1/D_i
* multiplicity m_i -> POSITIVE power  APd_i^m_i
```

- Internally a denominator becomes a symbol $\texttt{APd}_i = 1/\mathrm{D}_i$; multiplicity is a positive exponent $\texttt{APd}_i^{m_i}$, so all of Phase 1/2 is polynomial-like, exactly what `FORM` is fast at
- Over-consumption is just $\texttt{APd}_i^{-1} = \mathrm{D}_i$ back in the numerator, never a bare variable

$\Rightarrow$ At the end `ApartResolveDenominators` maps $1/\texttt{APd}_i \to \mathrm{D}_i$ and $\texttt{APd}_i^m \to \texttt{den}(\mathrm{D}_i)^m$

UNIVERSITY of DEBRECEN

Trick #2: per-term state via dollars

```
$Position = `AuxFuncID'($TMPden);
Inside $Position;
* look up the index in the global denominator pool:
  id `AuxFuncID'(`Sym1'?`DenSetID'[`Sym2']) = `Sym2';
EndInside;
* worker-local: each tform worker gets its own copy
ModuleOption,Local,$Position;
```

- Multiplicities, active set, pivots, adjugate entries: all carried in per-term dollars

$\Rightarrow$ `ModuleOption,Local` makes them thread-private, one copy per `tform` worker

UNIVERSITY of DEBRECEN

Read-only ctables (and the one exception)

- Built once at setup, read by every term thereafter:
  - `APcoeffTbl`: the denominator coefficient matrix C
  - `APdenTbl`: the denominator polynomials D_i
  - `APpermTbl`: the (optional) fill-reducing elimination order
- The cover-first null-relation cache `APnullTbl` is the exception: created `sparse`, it grows during Phase 1
- The cache-derivation step is wrapped in `NotInParallel`: run single-threaded so the table reads/writes do not race ...

$\Rightarrow$ ... while the per-term elimination that uses the cache runs in parallel as usual.

UNIVERSITY of DEBRECEN

Phase 1: two strategies for the null relations

Each term needs the null relations of its active denominator set. Two complementary paths:

- **Determinant branch** (`APuseGlobalNullRelations=0`): recompute relations per term by cofactor expansion. Cheap for few denominators; scales poorly when many terms share large active sets.
- **Cover-first cache** (`APuseGlobalNullRelations=1`): derive the full set's relations once, then for each active-set bitmask walk back to a cached superset and drop the absent denominators.

– Optional fraction-free Bareiss (`APuseBareiss`) for numeric coefficients

⇒ Both selected by a `#Redefine`; we benchmark both (later)

UNIVERSITY of DEBRECEN

Trick #3: the determinant via `distrib_`

```
#procedure ApartDeterminantThruCofactorExpansion(
      RowFuncID,ColFuncID,TableID,AuxFuncID,Sym1,...,Sym4)
* 1x1 base case (2x2 has its own direct rule, omitted):
  id `RowFuncID'(`Sym1'?)*`ColFuncID'(`Sym2'?) = `TableID'(`Sym1',`Sym2');
* larger n: expand the first row; distrib_'s -1 gives the signs
  Repeat;
    id `RowFuncID'(`Sym1'?,?a)*`ColFuncID'(?b) =
      `TableID'(`Sym1')*`RowFuncID'(?a)
      *distrib_(-1,1,`AuxFuncID',`ColFuncID',?b);
    id `TableID'(`Sym1'?)*`AuxFuncID'(`Sym2'?) = `TableID'(`Sym1',`Sym2');
  EndRepeat;
#endprocedure
```

- `distrib_` (a FORM built-in) generates the signed cofactor sum; its first argument -1 requests the permutation signs

$\Rightarrow$ Cofactor $O(n!)$ vs `Bareiss` $O(n^3)$ (`APuseBareiss`); cofactor wins for symbolic as each `Bareiss` pivot needs an exact polynomial division

UNIVERSITY of DEBRECEN

Trick #4: preprocessor-built dollar names

```
#procedure ApartAdjugateMatrix(
       dMat,SepChar,Dim,RowFuncID,ColFuncID,AuxFuncID,TableID,Sym1,...,Sym4)
  #Do iRow=1,`Dim'
    #Do jCol=1,`Dim'
      $`dMat'`iRow'`SepChar'`jCol' = $RowsAndColumns;
      Inside $`dMat'`iRow'`SepChar'`jCol';
        Multiply (-1)^(`iRow'+`jCol');
        Transform,`RowFuncID',dropargs(`jCol',`jCol');
        Transform,`ColFuncID',dropargs(`iRow',`iRow');
        #call ApartDeterminantThruCofactorExpansion(`RowFuncID',
              `ColFuncID',`TableID',`AuxFuncID',`Sym1',`Sym2',`Sym3',`Sym4')
      EndInside;
      ModuleOption,Local,$`dMat'`iRow'`SepChar'`jCol';
    #EndDo
  #EndDo
#endprocedure
```

⇒ **Name** built at preprocess time; `ModuleOption,Local` sits outside `Inside` (needs `FORM 5`)

UNIVERSITY of DEBRECEN

Trick #5: everything is a named argument

- Every symbol, function head, table and set a procedure touches is passed in, never hard-coded:

```
ApartAdjugateMatrix(dMat, SepChar, Dim, RowFuncID, ColFuncID,
                    AuxFuncID, TableID, Sym1, ..., Sym4)
```

- Consequences:
  - procedures compose: a caller reuses them on its own matrices/functions without name clashes
  - the same procedure runs from several call sites in one script with disjoint working state
  - unit tests instantiate each procedure with test-local names

⇒ House rule: no underscores in identifiers (the trailing _ is reserved for `FORM` built-ins like `distrib_`); CamelCase throughout

UNIVERSITY of DEBRECEN

Trick #6: scripting a $ with Inside, stashing to a Spectator

```
CreateSpectator SymDep "symdep.spec";
.sort
$tag = 1;
ModuleOption,Local,$tag;
#Do Sym={x,y}
  If (occurs(`Sym'));
    Inside $tag;
      If (count(dep,1)==0);
        Multiply dep(`Sym');
      Else;
        id dep(?a) = dep(?a,`Sym');
      EndIf;
    EndInside;
  EndIf;
#EndDo
Inside $tag;
  If (count(dep,1)>0) ToSpectator SymDep;
EndInside;
```

UNIVERSITY of DEBRECEN

Trick #6: how it works

- `CreateSpectator` opens a side expression you stream terms into; it stays in memory and spills to its file (`symdep.spec`) only once it grows too large
- `$tag` is `ModuleOption,Local` – one scratch dollar per term. `Inside $tag; … EndInside;` is the only way to run `id`/`Multiply`/`If` on a dollar's content: `FORM` treats the `$` as a one-term expression
- The loop builds `dep(x,y)` inside `$tag` – which active symbols the term contains

⇒ `ToSpectator` (from within `Inside $tag`) streams that tag into the spectator; `CopySpectator` harvests it back into a normal expression

UNIVERSITY of DEBRECEN

Trick #7: ChainIn + Transform AddArgs

Wrap each Basis factor in Aux, combine to turn $\prod_i \texttt{Basis}(a_i)$ into $\sum_i \texttt{Basis}(a_i)$, then unwrap. The **combine**, naively a repeat:

```
repeat id Aux(Sym1?)*Aux(Sym2?) = Aux(Sym1+Sym2);
```

Instead, FORM's chain machinery:

```
ChainIn Aux;                    * gather into one Aux(...)
Transform,Aux,AddArgs(1,last);  * sum the arguments
```

- The re-matching, re-sorting repeat loop becomes two loop-free passes: the win that matters on the heavy tail

$\Rightarrow$ Knowing FORM's Chain*/Transform vocabulary turns an inner-loop hotspot into straight-line code

UNIVERSITY of DEBRECEN

Trick #8: `Symmetrize` orders denominators & null relations

```
* bring the active denominator set into FORM's canonical order ...
Symmetrize Dens;
* ... the ordered argument tuple is then a stable cache key:
id Dens(?a) = Dens(?a)*NullTbl(?a);
```

- `Symmetrize` sorts a function's arguments into `FORM`'s canonical order; we use it to order denominator sets, basis columns and null-relation indices
- The ordered tuple is a stable signature: it lets the null-relation cache match a denominator set however it was assembled
  - It is `FORM`'s internal normal order, not the obvious one; sign-distinct factors in particular can flip it

$\Rightarrow$ Bonus: choices then come out reproducible, worker-to-worker

UNIVERSITY of DEBRECEN

A few FORM lessons learned

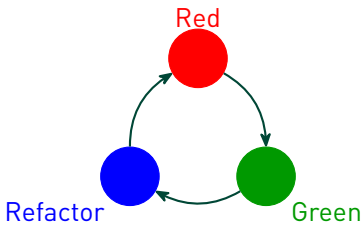
- Per-term dollars stream with the terms: each term sets the $ as it is processed, and the `.sort` freezes it to the last term's value, so set and use it in the same module
- Helpers under `Keep Brackets`: it screens the bracket content, so an `id` there cannot bind a wildcard to it; run such helpers in the driver, outside the brackets
- Canonical sort $\neq$ obvious order: FORM keeps terms in its internal normal form, so the occurrence an `id once` rewrites "first" may not be the one you expect (a sign on an argument can flip it)
- Reserved syntax: no `_` in names; dollar-valued `#call` arguments are passed bare (no $); watch whitespace in `#call` argument lists

UNIVERSITY of DEBRECEN

Testing & benchmarks

UNIVERSITY of DEBRECEN

Test-driven development with `check.rb`



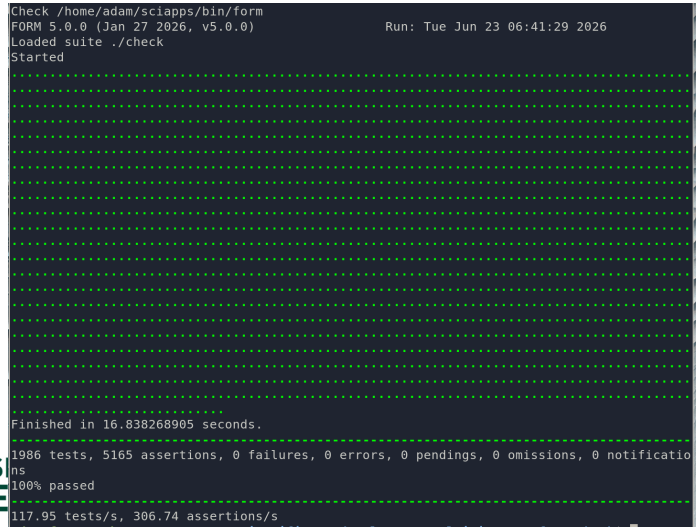
- Strictly modular: if a routine needs more than one sentence to describe, split it
- ∼50 procedures, each with its own `.frm` unit test (many cases per file)
- Driven by `check.rb` shipped with `FORM`: folds of code + Ruby assertions on `nterms`/results
- Plus end-to-end round-trip checks on the full driver

⇒ Algorithms are not final: the test net makes refactoring safe

UNIVERSITY of DEBRECEN

No regression: the whole test suite passes

1986 tests, **5165** assertions: 0 failures, 0 errors, 100% passed under `FORM 5.0.0`, in 16.8 s



Trick #9: a representation-independent check

Verify "input = output" numerically:

```
Local Foutput = Finput;
.sort
Hide Finput;      * driver acts on EVERY visible expression
.sort
#call ApartMultiLinApart(den,x,y)
.sort
Unhide Finput;
#call ApartNumericCheck(NumDiff,Finput,Foutput,
                        den,Vars,Params,n1)
* large random PRIMES for every variable & parameter:
* NumDiff = 0  <=>  algebraically equal (Schwartz-Zippel)
```

- Partial fractions have many valid forms; a numeric round-trip is ground truth

$\Rightarrow$ `Hide`/`Unhide` is essential: the driver is a whole-stream operator

UNIVERSITY of DEBRECEN

Trick #10: timing the phases from inside

```
#call ApartEliminateNullRelations()
.sort
#message PHASEMARK PHASE1END   * emitted in stream order
... Phase 2 ...
.sort
#message PHASEMARK PHASE2END
```

- Run `tform -w16 -W` so each `.sort` prints a `WTime` (wall-clock) line; difference the cumulative `WTime` at consecutive markers ⇒ per-phase wall time
- Gotcha: without `-W` you only get master-thread CPU `Time`, which under `-w16` under-represents (even inverts) the Phase 1/Phase 2 split

⇒ A few `#message` lines turn the run log into a phase profiler

UNIVERSITY of DEBRECEN

Benchmark: the double-pentagon IBP matrix

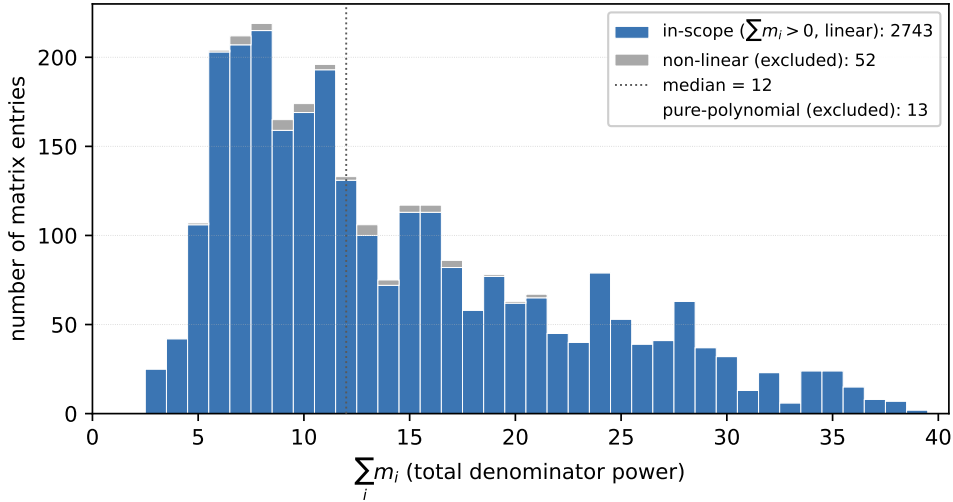
- Two-loop non-planar five-point double-pentagon dlog-basis IBP coefficient matrix [Bendle et al., arXiv:1908.04301; also used by Böhm et al. (arXiv:2008.13194) and `MultivariateApart`: Heller, von Manteuffel (arXiv:2101.08283)]
- A 26×108 matrix of rational functions of $\{\varepsilon, s_{15}, s_{23}, s_{34}, s_{45}\}$; active variables $\{\varepsilon, c_2, c_3, c_4, c_5\}$
- 2743 in-scope cells: linear-in-active-variable denominators (excludes Gram-determinant and pure-polynomial entries)
- **Numerator set to** 1 for the timing, to isolate the partial-fraction work itself (not the cell's numerator)

$\Rightarrow$ A continuous range of input weights $\sum_i m_i$ from very small to moderately heavy: a realistic stress test

UNIVERSITY of DEBRECEN

Benchmark: distribution of input weights

2743 in-scope cells, $\sum_i m_i$ from 3 to 39 (52 Gram + 13 pure-polynomial excluded)



Benchmark: determinant vs. cover-first cache

Full in-scope population (2743 cells), `tform` 16 workers, AMD Ryzen 9 5900XT, 128 GB RAM:

mode	Σ wall [s]	max [s]
determinant	103 208	3 027
cache	82 900	2 544

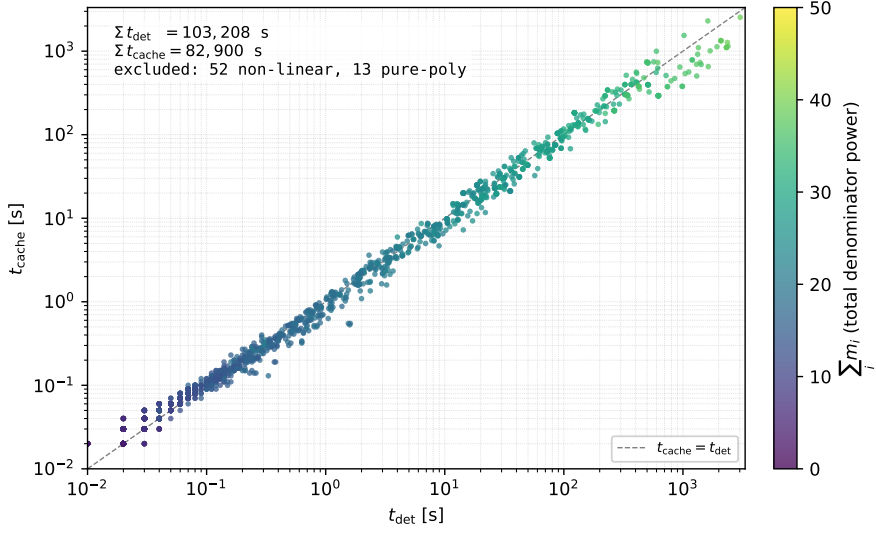
- Cover-first cache saves $\sim$20% of total wall time
- The heavy tail dominates: the 122 cells with $\sum_i m_i \geq 31$ are $\sim$80% of the total, and that is where the cache pulls ahead
- `tform` multicore scaling carries the whole sweep; the heavy cells are CPU-bound and parallelize well across the 16 workers

$\Rightarrow$ We recommend the cache branch; keep determinant as a fallback

UNIVERSITY of DEBRECEN

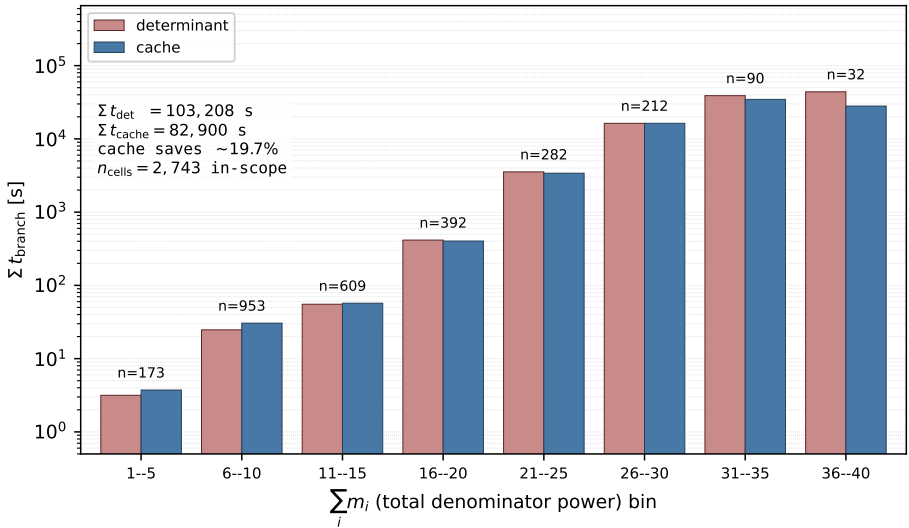
Benchmark: per-cell, cache vs. determinant

Each cell is a point (color = $\sum_i m_i$); below the y = x diagonal the cache wins



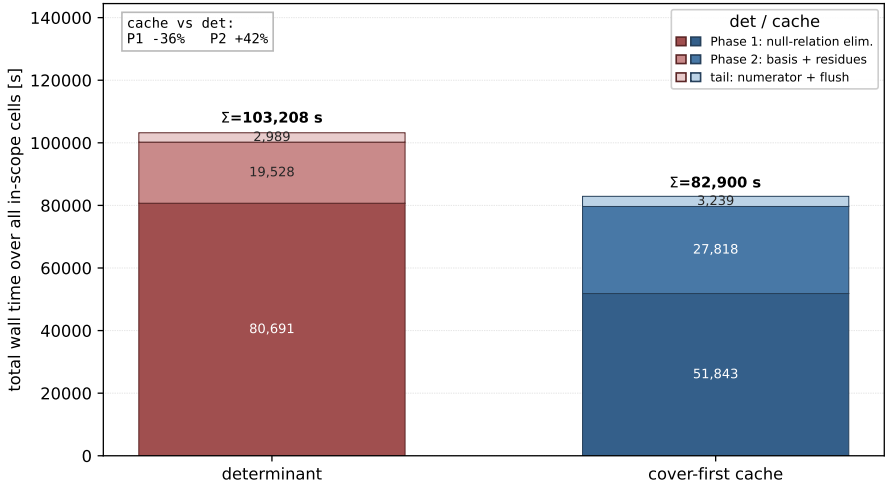
Benchmark: where the saving comes from

Per-bin total wall, determinant (red) vs. cache (blue); the $\sum_i m_i \geq 31$ tail carries the saving



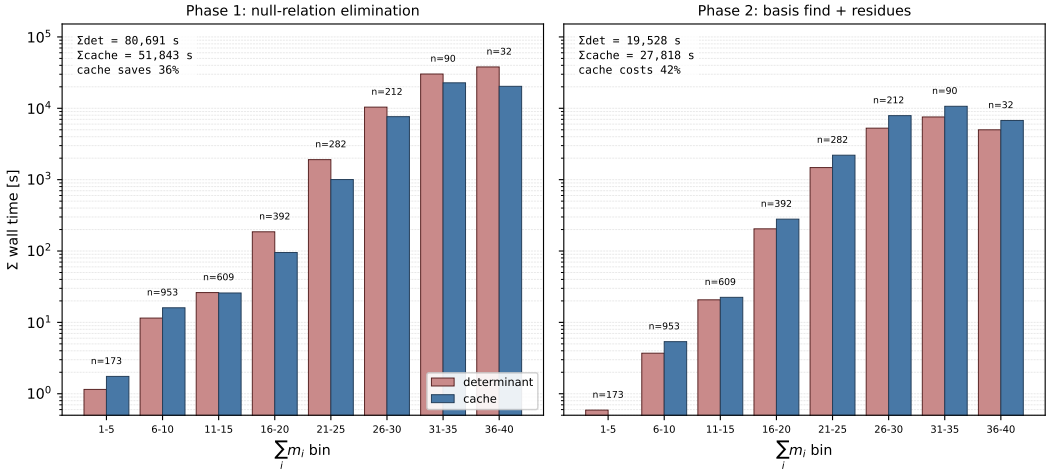
Benchmark: where the time goes (Phase 1 vs Phase 2)

`PHASEMARK+-W` split the wall: cache's win is all in Phase 1, it even loses Phase 2



Benchmark: the Phase 1 / Phase 2 trade-off

Cache wins Phase 1 on the heavy tail and loses Phase 2; Phase 1 dominates the net



Usage: a complete program

```
#include formapart.h
* mode: 0 = determinant, 1 = cache
#Redefine APuseGlobalNullRelations "1"

CFunction den;
Symbols x, y;
Local F = den(x)^2*den(y)
          *den(x+y)*den(x+y-1);
.sort
#call ApartMultiLinApart(den,x,y)
Print +s;
.end
```

- Denominators wrapped in one user cfunction (`den`), multiplicities as exponents
- Active variables follow the wrapper name in the `#call`; everything else (masses, ε, invariants) is a spectator, untouched
- `den` arguments must be linear in the active variables (user's responsibility)

$\Rightarrow$ Fill-reducing order: `APglobalOrder`, `APorderDescending`

Worked example: input $\to$ output

Input $\mathsf{f(x,y)} = \dfrac{1}{\mathsf{x}^2\,\mathsf{y}\,(\mathsf{x}+\mathsf{y})\,(\mathsf{x}+\mathsf{y}-1)}$ gives

```
F =
  + den(x)*den(-1+y+x)
  + den(x)^2*den(-1+y+x)
  - den(x)^2*den(y)
  - den(x)^2*den(y)^2
  - den(x)*den(y)
  + den(x)*den(y)^3
  + den(y)*den(-1+y+x)
  - den(y)^3*den(y+x);
```

- Reading `den(a)` as $1/\mathsf{a}$: an eight-term decomposition
- Every term has $\leq \mathsf{n} = 2$ distinct denominators, all from the original set
- No spurious singularities: the guarantee holds term by term

$\Rightarrow$ Matches the Mathematica implementation's choice; both pass the numeric round-trip

UNIVERSITY of DEBRECEN

Summary

- `LinApart3`: multivariate partial fractioning, no spurious singularities, native in `FORM`
- Term-stream design: read-only ctables + per-term dollars, parallel under `tform`
- $\sim$50 unit-tested procedures; `FORM 5` dollar placement is load-bearing
- Two Phase-1 branches; cache wins $\sim$20% on the double-pentagon

Thank you! Questions and `FORM` tricks welcome

UNIVERSITY of DEBRECEN