FORM for Computer Algebra in Particle Physics

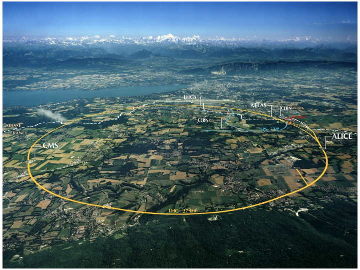
FLINT Development Workshop

Josh Davies

UNIVERSITY OF LIVERPOOL

28th October, 2025

Why Particle Physics?



What is the Universe made of?

- what are the building blocks of matter, how do they interact?
- do the known particles behave as the *Standard Model* predicts?
- is there *Beyond the Standard Model* physics?
- how to explain neutrino oscillations, Baryon asymmetry?
- what is *dark matter*?
- can we reconcile the SM with Gravity?
- ...

The *Large Hadron Collider* is our energy-frontier machine which aims to answer these questions.

- it collides high energy protons, and measures the results
- collaboration between experimental and theoretical physicists

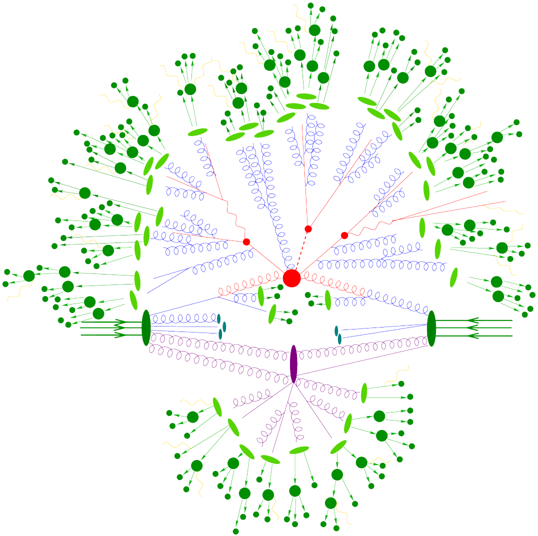
LHC Particle Interactions

Hadron collisions are very complicated!

- Parton Distribution Functions
- Hard interaction $\leftarrow$ want to measure this
- Parton shower
- Hadronization
- Hadron decays
- Secondary interaction

We need to make theoretical predictions for all of these processes, which can be compared with our experimental measurements.

Predicting hard interaction scattering rates is the computer-algebra intensive part.



[Gleisberg, Höche et al. ‘08]

Hard Interactions: Perturbative Approach

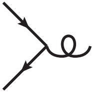
Starting from a **Lagrangian**, we derive *Feynman Rules* for a perturbative description

- assume **couplings** are small
- define how particle fields propagate (edges) and interact (vertices)

$$\mathcal{L}_{QCD} = -\frac{1}{4} F^a_{\mu\nu} F_a^{\mu\nu} + \bar{\psi}_i \left(i \not{D} - m \right)_{ij} \psi_j$$

$$\downarrow$$

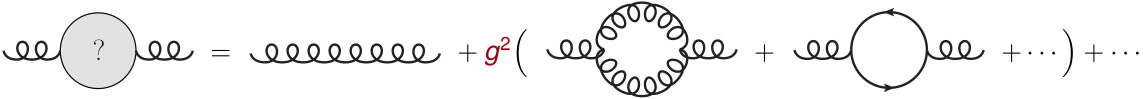
 $: \quad -g\, f^{abc} \left(g^{\mu\nu} [k_1 - k_2]^\rho + g^{\nu\rho} [k_2 - k_3]^\mu + g^{\rho\mu} [k_3 - k_1]^\nu \right),$

 $: \quad -i\, g\, \gamma^\mu (t^a)^j_i,$

 $: \quad \frac{-i\, \delta^{ab}}{k^2} \left(g^{\mu\nu} + [\xi - 1] \frac{k^\mu k^\nu}{k^2} \right),$

 $: \quad \frac{i\, \delta^{ij}}{\not{p} - m},$ ...

Using these "puzzle pieces", we can describe particle interactions:



Computing Scattering Amplitudes

Using the Feynman rules, we can generate scattering amplitudes up to some *perturbative order*.

The standard set of “straightforward steps”:

1. Generate appropriate diagrams
2. Insert Feynman rules
3. Simplify/process/compute
4. Integration-by-Parts Reduction of loop integrals
5. Compute “Master Integrals”

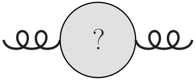
In practice, these steps are extremely challenging, both computationally and mathematically.

Many bespoke software packages have been developed for such computations:

- Graph generators: **`qgraf`**, **`FORM`**, **`FeynGraph`**
- Computer algebra: **`FORM`**, various packages for **`Mathematica`**, **`Symbolica`**
- IBP reduction: **`FIRE`**, **`Kira`**, **`LiteRed`**, ...
- Numerical integration: **`pySecDec`**, **`FIESTA`**

The Need for Computer Algebra

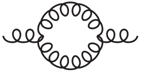
The number of graphs/diagrams grows as a factorial with the perturbative order.



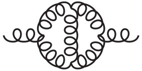
$$= (1 \text{ dia}) + g^2(4 \text{ dia}) + g^4(26 \text{ dia}) + g^6(473 \text{ dia}) + g^8(12\text{K dia}) + g^{10}(381\text{K dia}) \cdots$$

The complexity of *each diagram* grows with the perturbative order.

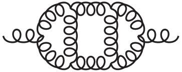
- more loops:



Feynman rules generate 144 terms ($\rightarrow$ 40 after merges)



Feynman rules generate 41K terms ($\rightarrow$ 2833 after merges)



Feynman rules generate 12M terms ($\rightarrow$ 279K after merges)

- more external particles: increasingly multivariate coefficients, complicated integrals

Integration-by-Parts Reduction

Scattering amplitudes depend on "loop integrals" over the internally unconstrained momenta:

- difficult to compute, but fortunately these integrals are not all independent
- expose linear relations between them via "integration-by-parts identities"
- solve these relations to reduce to a basis set of "master integrals"

Conceptually easy but technically difficult:

- set of (potentially) millions of integrals $\longrightarrow$ more millions of (sparse) linear relations
- coefficients are multivariate polynomials
- requires large systems with lots (multi-TB) of RAM

We have dedicated software to solve this problem as efficiently as we can manage:

- **`FIRE`**, **`Kira`**, ... various others
  - not Computer Algebra Systems, but rely on high-performance polynomial arithmetic
  - **`FIRE`** uses **`FLINT`** for poly arithmetic, **`Kira`** for finite-field arithmetic
  - (both have previously relied/rely on **`Fermat`** for poly arithmetic)
- "Recent" (in our field) optimization: sample the solution over finite fields, reconstruct
  - helps for some problems, but not all

An (Incomplete) History of Computer Algebra in Particle Physics

`Schoonschip` (1963) Veltman: First CAS.

- For CDC and Motorola 68000. Landmark computations involving 50K terms.

Then a lot of development, for e.g.:

- **`Macsyma`**, **`REDUCE`** (1968), **`SMP`** (1981), **`Maple`** (1982), **`Fermat`** (1985), **`Mathematica`** (1988)

`FORM` (Jos Vermaseren) designed as a spiritual, portable (**`C`**), successor to **`Schoonschip`** for HEP:

- **`FORM 1`** (1989), (work started in 1984)
- **`FORM 2`** (1991), commercial package requiring license and fee
- **`FORM 3`** (2000), free (gratis), early parallelisation implementations (MPI)
- **`FORM 3.3`** (2010), free (libre GPLv3), **`pthreads`** parallelisation, **`GMP`**
- **`FORM 4`** (2012), polynomial arithmetic routines
- **`FORM 4.1`** (2013), expression optimization routines
- **`FORM 4.2`** (2017), features for a particular package (**`FORCER`**)
- **`FORM 4.3`** (2022), mostly bug fixes
- **`FORM 5`** (**2025**), diagram generator, floating-point coeff. mode, **`FLINT`** interface

Since 2023, annual “Developers’ Workshops” to discuss future direction, feature requests, etc.

- effort to engage wider community in development since the retirement of Vermaseren in 2018

Why FORM?

FORM was developed specifically with the needs of high-energy physics computations in mind:

- processing of enormous expressions, not limited RAM (potentially $\sim$10TB...)
- efficient handling of Dirac algebra
- optimizing large expressions for fast numerical evaluation in compiled code
- easy-to-use parallelization
- free and open source (since 2000, 2010) [https://github.com/form-dev/form]

Vast majority of cutting-edge multi-loop computations have used **FORM**, including

- four and five loop QCD beta function
- three (and four, partially) loop Splitting Functions for PDF evolution
- three loop quark and gluon form factors
- many two loop $2 \to 2$ scattering amplitudes
- ...

FORM papers have over 3K citations. Papers which cite **FORM** have $\sim$140K citations.

It has had an enormous impact on our ability to analyse LHC data.

An example FORM script

```
#-
Symbol x,y,z;
CFunction f,g;

Local test = (x+y)^3 + f(1,2,y,3,y,4)
   + f(1,x,2,3) + g(1,1);

Identify y = z-x;
Identify f?(?a,x?,?b,x?,?c) = g(?a,?b,?c);

Print;
.sort

Argument g;
   Multiply 3;
EndArgument;

If ( (Count(z,1) > 0) || (Match(f?(?a,12,?b))) );
   Multiply 2;
EndIf;

Print;
.end
```

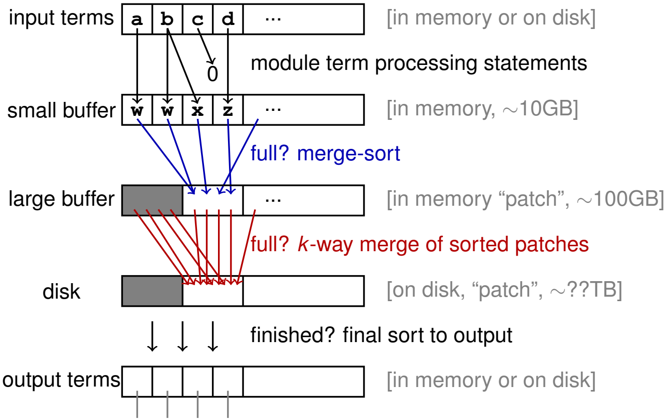
```
FORM 5.0.0-beta.1 (Oct 21 2025, v5.0.0-beta.1-255-
     g9b7e97d)  Run: Mon Oct 27 13:53:06 2025

    #-

Time =         0.00 sec    Generated terms =         13
            test           Terms in output =          4
                           Bytes used      =        184

   test =
      z^3 + f(1,x,2,3) + g + g(1,2,3,4);


Time =         0.00 sec    Generated terms =          4
            test           Terms in output =          4
                           Bytes used      =        184

   test =
      2*z^3 + f(1,x,2,3) + g + 2*g(3,6,9,12);

  0.00 sec out of 0.00 sec
```

- All vars defined + typed
- Everything is always expanded
- Patterns are term-local
- Wildcard type meaningful
- **x?** : wildcard (MMA **x_**)
- **?a** : arg field (MMA **a__**)

Term processing and sorting system

FORM's multi-level sorting of huge expressions makes it uniquely suited for our computations.

- computer memory has grown over the years, but so has the size of our calculations
- each "module" is processed with the following structure:




```
Symbol a,b,c,d,w,x,y,z;
Local test = a + b + c + d ...
Identify a = w;
Identify b = w + x;
Identify c = 0;
Identify d = z;
.sort
```

Term-by-term processing as "depth-first tree".

This design enables this sorting procedure, but limits pattern matching flexibility.

$k > k_{\max}$? disk→disk patch merge.

Disk patches are compressed (**zlib, zstd**)

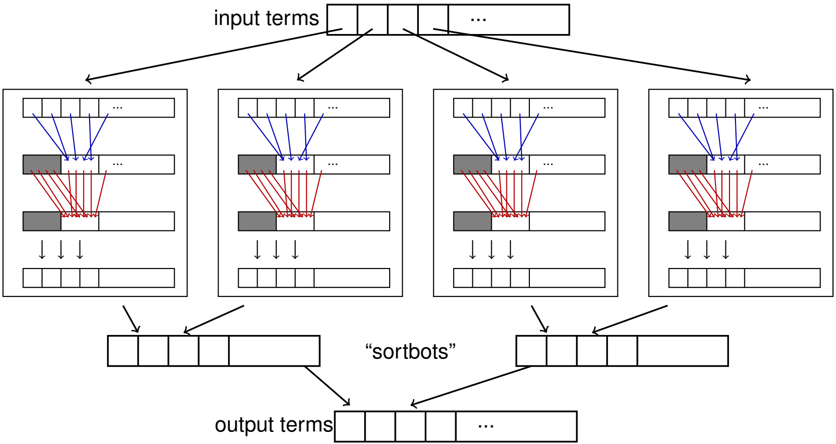
Buffer sizes are all user-configurable.

The output becomes the next module's input.

Parallelization in FORM

Parallelization in **TFORM** (pthreads) and **ParFORM** (MPI) (deprecated)

- share (batches of) input terms between workers which process and sort in parallel
- final (parallel binary) merge of sorted results from each worker
- (almost) no modification of user scripts required! Very easy to use.



Polynomial Operations in FORM

Built-in functions, which use built-in polynomial code (∼2012)

- **gcd_**, **div_**, **rem_**, **mul_**, **inverse_**
- Factorization of expressions, function arguments, dollar variables
- **PolyRatFun** : rational-polynomial term coefficients

```
#-
Symbol x,y,z,n;
CFunction rat,num;
PolyRatFun rat;

Local test = (x+y+z/y)^3
   + num(gcd_((x+y)^2*(x-y), (x+y)^3));

Identify many x?!{z}^n? = rat(x^n,1);

FactArg num;
ChainOut num;
SplitArg num;

Print +s;
.end
```

```
Time =        0.00 sec    Generated terms =       11
             test         Terms in output =        5
                          Bytes used      =      688

   test =
       + z*rat(3*x^2 + 6*x*y + 3*y^2,y)
       + z^2*rat(3*x + 3*y,y^2)
       + z^3*rat(1,y^3)
       + rat(x^3 + 3*x^2*y + 3*x*y^2 + y^3,1)
       + num(y,x)^2*rat(1,1)
      ;
```

Interface with FLINT

FORM 5 features an interface to **FLINT**, for faster polynomial arithmetic.

- Requires **FLINT** ≥v3.2.0, need the fixes for:
  - [FLINT #1652] (3.1) (reentrancy of `gr_method_tab_init`)
  - [FLINT #1998] (3.2) (bug in `fmpz_mpoly_factor`)
- Enabled by default if **FLINT** is found at compile time. Fallback to built-in code.
- Implements all but full expression factorization, modular arithmetic modes. **TODO!**

The role of the interface code:

1. translate **FORM**-internal representation to **FLINT** `fmpz`, `fmpz_poly`, `fmpz_mpoly`

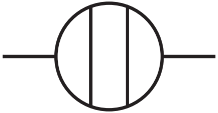
```
-12345 * a^2 * b * c^0 * d^3 — 12 1 8 20 2 21 1 23 3 12345 1 -3
                              ↓
fmpz_mpoly_push_term_fmpz_ui(arg, -12345, {2,1,0,3}, ctx);
```

2. use **FLINT** `poly` or `mpoly` routines for computation
   - `fmpz_mpoly_gcd`, `fmpz_mpoly_mul`, `fmpz_mpoly_quasidivrem`, ...
3. translate back to **FORM** representation, re-sort terms in **FORM** ordering

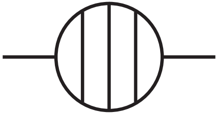
Performance

The **FLINT** routines have excellent performance, particularly for multivariate polynomials.

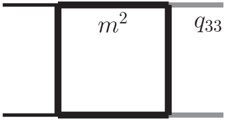
- **`minceex`** DIS moment test, **`ep`**-exact, univariate
  - $N = 8$: **40s** $\rightarrow$ **34s (1.2x)**
  - $N = 10$: **146s** $\rightarrow$ **123s (1.2x)**
  - $N = 12$: **538s** $\rightarrow$ **460s (1.2x)**



- **`forcer`** IBP test, **`ep`**-exact, univariate
  - 16 prop: **210s** $\rightarrow$ **116s (1.8x)**
  - 17 prop: **583s** $\rightarrow$ **287s (2x)**
  - 18 prop: **1673s** $\rightarrow$ **873s (1.9x)**



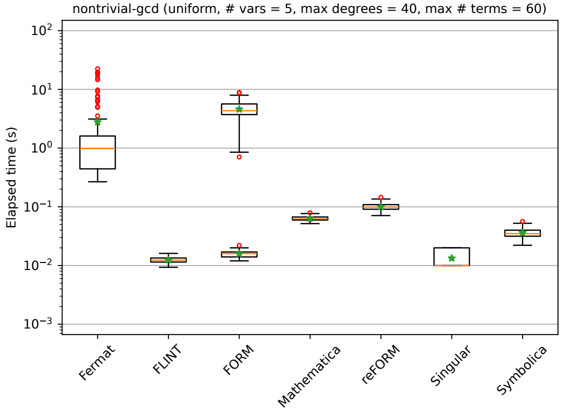
- **`mbox1l`** IBP (1L box, vars $d, q_{12}, q_{13}, q_{33}, m^2$) multivariate
  - **`mbox1l(2,2,2,1)`**: **0.97s** $\rightarrow$ **0.26s (3.7x)**
  - **`mbox1l(3,2,2,2)`**: **18.4s** $\rightarrow$ **1.18s (16x)**
  - **`mbox1l(3,3,2,2)`**: **76.3s** $\rightarrow$ **2.58s (30x)**
  - **`mbox1l(3,3,3,3)`**: **514s** $\rightarrow$ **10.3s (50x)**

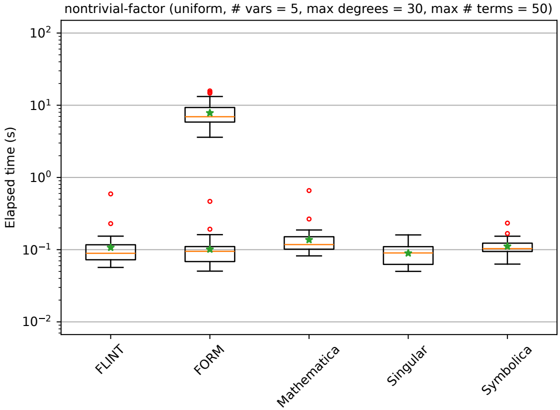


[Single-thread reduction of **`mbox1l(3,3,2,2)`**: 12.2s (**`FORM`**), 329s (**`MMA 14.2, LiteRed v1.84`**)]

Performance (II)

[Takahiro Ueda's `polybench`] **(∼100x, depending on settings...)**





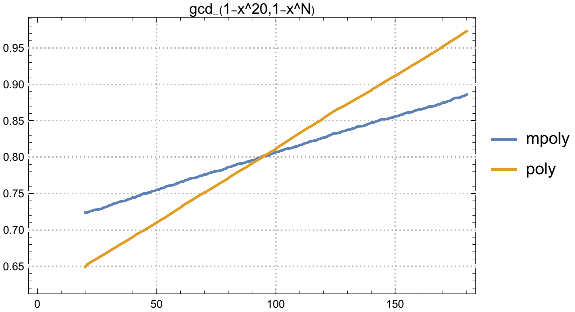
Performance (III)

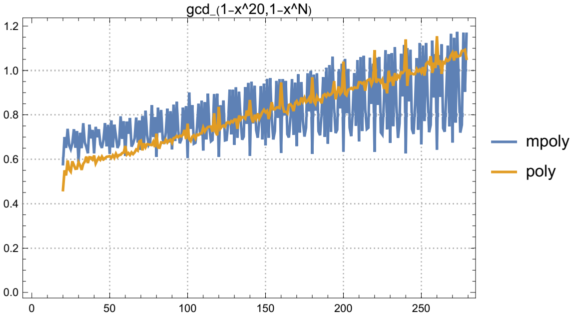
There are edge cases to deal with, e.g.: `gcd_(1-x^20,1-x^10000000)`

- `Off flint;` 60ms
- `On flint;` 249ms
- `On flint;` 13ms (but force use of `mpoly`)

This is because `fmpz_poly` is dense, `fmpz_mpoly` is sparse (and `FORM` built-in is sparse).

- investigating adding a "density" heuristic: (num. terms)/(max degree) < 0.02 ? Use `mpoly`.





Input like this doesn't typically occur in "real physics calculations", anyway.

Conclusions

FORM has been one of our most important software packages for decades, and will continue to be!

- used directly for computation, by many people
- used by a variety of packages
- new packages are still being actively developed which use **FORM**

FORM is not averse to using external libraries:

- **GMP**, **MPFR**, **zlib**, **zstd**, and now **FLINT**
- excellent way to incorporate effort and expertise from other fields!

There has been a lot of development in the last few years,

- (despite retirement of Vermaseren and lack of financial support)
- Developers' Workshops have been very effective