Adding FUEL to FIRE for Faster IBP

Mao Zeng, Higgs Centre for Theoretical Physics, University of Edinburgh

Talk at MathemAmplitudes 2023: QFT at the Computational Frontier, University of Padova, 26 Sep 2023

- Kirill Mokrov, Alexander Smirnov, MZ, arXiv:2304.13418
- Alexander Smirnov, MZ, in progress

Outline

- Background – IBP reduction
- Rational function simplification in Laporta algorithm
  - Choice of computer algebra systems (simplifiers)
  - Performance requirements
- Benchmark results

Background

- Integration-by-parts (IBP) reduction [Chetyrkin, Tkachov, '81] is ubiquitous in modern Feynman integral calculations.
- A family of integrals parametrized by powers of propagators and irreducible scalar products (ISPs)

$$I_{a_1,a_2,\ldots,a_n} = \int \left(\prod_{i=1}^{L} d^d l_i \right) \rho_1^{-a_1} \rho_2^{-a_2} \ldots \rho_n^{-a_n}$$

- Total derivatives integrate to zero in dimensional regularization $\Rightarrow$ linear relations between above integrals.

Laporta algorithm

- **Solves large linear system** to express complicated integrals in terms of simple integrals, under some ordering. [Laporta, '01]
- **Codes:** AIR, Reduze, LiteRed, FIRE, Kira, FiniteFlow, Blade, NeatIBP...
- **Alternatives:** symbolic reduction rules, intersection theory, Groebner bases, D modules… Or not doing IBP at all (SecDec, LTD, FeynTrop...)
- **Optimizations & Variations:** ordering / pivoting for equations and variables, trimming IBP equations by Lie algebra, syzygy equations & numerical unitairty, finite fields & reconstruction, choosing "improved" master basis, block triangular form...

FIRE

- IBP program developed over many years [A.V. Smirnov, ’08. A.V. Smirnov, V.A. Smirnov, ’13. A.V, Smirnov, 14. A.V. Smirnov, F.S. Chukharev, ’19]
- Implements Laporta algorithm. Can use symmetry & reduction rules from LiteRed [R.N. Lee, ’12]. Initially written in Mathematica, available in C++ since version 5. Supports modular arithmetic, MPI in version 6.
- Trims IBP equations by Lie algebra. [R.N. Lee, ’08] Forward reduction w/ tail masking [Anastasiou, Lazapoulos, ‘’04], then backward substitution.
- Until our work, uses Fermat via gateToFermat library by M. Tentukov.
- Applied to many cutting-edge loop calculations. Recently used in 4-loop classical electrodynamics [Bern, Herrmann, Roiban, Ruf, Smirnov, Smirnov, MZ, ’23]

Coefficient simplification in FIRE

- During IBP calculation, FIRE (C++ version) needs ***external help*** in simplifying expressions of the form

$$\sum_k \frac{\frac{\text{Poly}_{k,1}}{\text{Poly}_{k,2}} \cdot \frac{\text{Poly}_{k,3}}{\text{Poly}_{k,4}}}{\frac{\text{Poly}_{k,5}}{\text{Poly}_{k,6}}}$$

→ *Polynomial in Horner form $a+x(b+x(c+dx))$*

or expanded form $a+bx+cx^2+dx^3$

- FIRE assembles the expression as a string (text blob), sends it to an external computer algebra system, or ***simplifier*** for short.
- The simplifier ***parses*** the expression into an internal representation, simplifies it (GCD computations etc.), and ***prints*** out a new string.

Choice of computer algebra system

- Combine terms into one simplified fraction.
  E.g. Mathematica: `Together[r]` Maple: `normal(r, expanded)`
- Default choice in FIRE, Reduze, Kira so far: ***Fermat*** by Robert Lewis. Served our community extremely well, but more actively developed and funded alternatives now exist…
- ***Let's explore options.*** Our new C++ library FUEL interfaces with CoCoA, Fermat, *FLINT*, FORM, GiNaC, Macaulay2, Maple, Maxima, Nemo, Pari/GP, *Symbolica* (FORM "successor"), Mathematica.
- Other relevant applications: transforming DEs to caonical form.

Special thank to Ben Ruijl for tirelessly customizing Symbolica for us.

FUEL: one interface for all simplifers

[Kirill Mokrov, Alexander Smirnov, MZ, arXiv:2304.13418 + Work in progress]

- **F**ractional **U**niversal **E**valuation **L**ibrary. Choose any simplifier, e.g.

```
fuel::setLibrary(“maple”);
```

- Declare the list of variables, and start simplifying!

```
std::vector<std::string> variables = {“d”, “s”, “t”};
fuel::initialize(variables);
fuel::simplify(“(d-3)*(s-t) + t^2/s”, thread_number);
```

- Returned result: `“(d*s^2-d*s*t-3*s^2+3*s*t+t^2)/s”`

- Technicality: communication via either pipes or C++ library.

Best simplifier is not obvious...

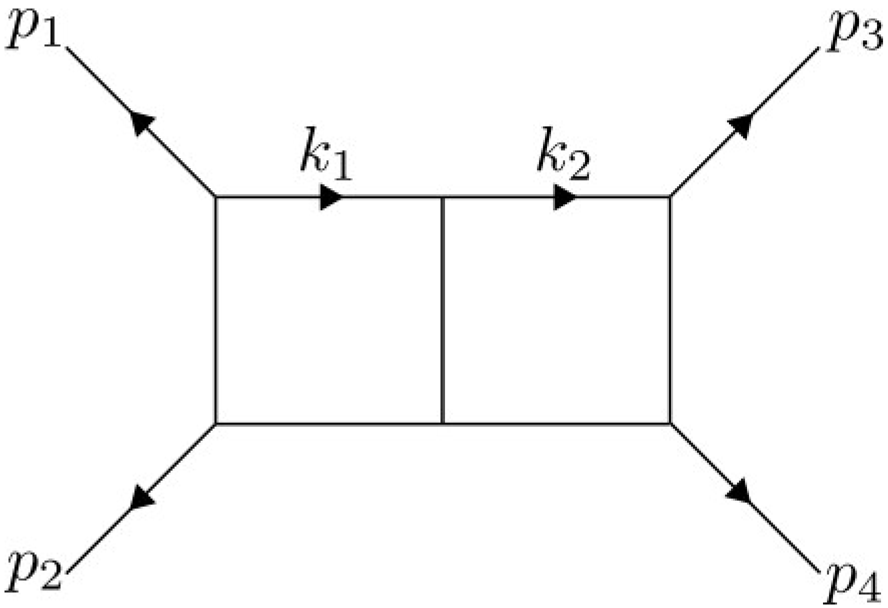
- How long does it take to simplify this expression through FUEL?

$$x = \frac{(a+b+c+d+f+g)^{14}+3}{(2a+b+c+d+f+g)^{14}+4} - \frac{(3a+b+c+d+f+g)^{14}+5}{(4a+b+c+d+f+g)^{14}+6}$$

- **Maple 2022:** 7.9 s
  Fermat 5.17: 98 s
  Mathematica 13.0: 169 s
- Naively, Maple is a superb choice (for FIRE etc.), and recent versions of Mathematica are not far from Fermat?
- The picture is different when we test a different problem.

Double box example

- Reduce below integral to 12 master integrals. ~15 s FIRE run sends ~0.5M expressions to simplifier (Femat), average ~30 μs turnaround



$$\times\ (k_2 + p_1)^2 (k_1 - p_3)^2$$

Double box: forward / backward runs

- Top level

```
G[1,{1,1,1,1,1,1,1,-1,-1}] -> (1)/(-(d-4)) G[1,{395,194}]
+ ((((3*d-18)*s^2)*t)/(4*d-20))/(-(d-4)) G[1,{1,1,1,1,1,1,2,0,0}],

G[1,{395,194}] -> (1)/(-(-1)) G[1,{1,1,0,1,1,1,1,0,-1}] +
                  (-1)/(-(-1)) G[1,{1,1,0,1,1,1,2,-1,-1}] … +
                  (1/(4*s^6))/(-(-1)) G[1,{395,193}]
```

Double box: forward / backward runs

- Top level

desired integral *“virtual” masked integral*

```
G[1,{1,1,1,1,1,1,1,-1,-1}] -> (1)/(-(d-4)) G[1,{395,194}]
+ ((((3*d-18)*s^2)*t)/(4*d-20))/(-(d-4)) G[1,{1,1,1,1,1,1,2,0,0}],
```

top-level master integral

```
G[1,{395,194}] -> (1)/(-(-1)) G[1,{1,1,0,1,1,1,1,0,-1}] +
                  (-1)/(-(-1)) G[1,{1,1,0,1,1,1,2,-1,-1}] … +
                  (1/(4*s^6))/(-(-1)) G[1,{395,193}]
```

- RHS integrals reduced by lower-sector runs until bottom sector (sunset). Then backward substitution from bottom up.

Double box: example expressions

- Sent to simplifier (Fermat etc.) near the beginning:

```
((s) * (-1)) - ((1) * (t))
((-2*s^2) * (- ((1) * (-s)))) - ((- ((1) * (-s))) * (-2*s^2))
((-2*s^2) * ((s) * (1))) - ((- ((1) * (-s))) * (- ((s) * (s))))
((-2*s^2) * ((s) * (-s))) - ((- ((1) * (-s))) * (- ((s) * (-s^2))))
((-2*s^2) * ((s) * (t))) - ((- ((1) * (-s))) * (- ((s) * ((s)*t))))
((-2*s^2) * (- ((1) * (-s)))) - ((- ((1) * (-s))) * (-2*s^2))
(-2*s^2) * (-s)
- ((- ((1) * (-s))) * (s^2))
- ((- ((1) * (-s))) * (s^2))
((-2*s^2) * (-t)) - ((- ((1) * (-s))) * ((2*s)*t))
((-2*s^2) * (t)) - ((- ((1) * (-s))) * ((-2*s)*t-s^2))
((-2*s^2) * (-t)) - ((- ((1) * (-s))) * ((2*s)*t))
((-2*s^2) * (t+s)) - ((- ((1) * (-s))) * ((-2*s)*t-s^2))
((-2*s^2) * (- ((1) * (t)))) - ((- ((1) * (-s))) * ((2*s)*t+s^2))
((-2*s^2) * ((s) * (-1))) - ((- ((1) * (-s))) * (- ((s) * (-s))))
((-2*s^2) * (t-s)) - ((- ((1) * (-s))) * ((-s)*t+s^2))
((-2*s^2) * (- ((1) * (-t)))) - ((- ((1) * (-s))) * ((-s)*t+s^2))
```

Double box: example expressions

- In the middle:

```
293856 0+((-s) * (- ((- ((-s) * (s))) * (- ((2*s^2) * (- ((-s) * ((s)*t
       ))))))))*(-1)
293857 0+((-s) * (- ((- ((-s) * (s))) * (- ((2*s^2) * (- ((-s) * (t))))
       ))))*(-1)
293858 0+(- ((((((2*s) * (s)) * (- ((-s) * (s)))) * (- ((-s) * (-s^2))))
        * (-1)))*(1)
293859 (2*d)*s^7+(2*s^7)*(0) +(-4*s^7)*(2) +((-4*s^6)*t-2*s^7)*(1) +(-2
       *s^7)*(1) +((4*s^6)*t-2*s^7)*(0) +((-4*s^6)*t-4*s^7)*(0) +((4*s^
       6)*t)*(-1)
293860 0+((-s) * ((((2*s) * (s)) * (- ((-s) * (s)))) * (2*s^2)))*(1)
293861 0+((-s) * (- ((- ((1) * (- ((-s) * (s))))) * (- ((2*s^2) * (- ((
       -s) * (s)))))))))*(-1)
293862 0+(2*s^7)*(1)
293863 0+((-s) * ((-2*s^5)*t+2*s^6))*(-1)
293864 0+(-2*s^7)*(1)
293865 0+((-s) * ((4*s^5)*t+2*s^6))*(1)
293866 0+((-s) * (- ((- ((1) * (- ((-s) * (s))))) * (- ((2*s^2) * (- ((
       -s) * ((s)*t))))))))*(-1)
```

Double box: example expressions

- Near the end:

```
       )
587709 -(((405*d^5-7335*d^4+52434*d^3-184882*d^2+321628*d-220960)*t+(-1
       62*d^5+3078*d^4-22986*d^3+84330*d^2-152100*d+108000)*s)/(((4*d^4
       -72*d^3+484*d^2-1440*d+1600)*s^2)*t))/(d-4)
587710 -(((9*d-36)*t^2+((16*d-64)*s)*t+(9*d-36)*s^2)/((2*s)*t))/(d-4)
587711 -(((405*d^5-7335*d^4+52434*d^3-184882*d^2+321628*d-220960)*t+(-1
       62*d^5+3078*d^4-22986*d^3+84330*d^2-152100*d+108000)*s)/(((4*d^4
       -72*d^3+484*d^2-1440*d+1600)*s^2)*t))/(d-4)
587712 -(((9*d-36)*t^2+((16*d-64)*s)*t+(9*d-36)*s^2)/((2*s)*t))/(d-4)
587713 -(((399*d^5-8425*d^4+70372*d^3-290416*d^2+591696*d-475920)*t+(-2
       16*d^5+4392*d^4-35352*d^3+140760*d^2-277200*d+216000)*s)/(((8*d^
       4-160*d^3+1192*d^2-3920*d+4800)*s)*t))/(d-4)
587714 -(((-4*d^2+36*d-72)*t+(18*d^2-126*d+216)*s)/((d-6)*t))/(d-4)
587715 -(((4*d^3-44*d^2+156*d-180)*t+(16*d^3-168*d^2+572*d-636)*s)/((d^
       2-9*d+20)*s^2))/(d-4)
587716 -((((-2*d+8)*s)*t+(-9*d+42)*s^2)/4)/(d-4)
587717 -((((3*d-18)*s^2)*t)/(4*d-20))/(d-4)
```

Performance for short expressions

- Example FIRE expression during 2-loop double box IBP run:
  `-((d-5)*s)/(-s)*(d-5)-(s)/(-s)*(d-5)`
- Test: simplify 10,000 times via FUEL (above string in, simplified string out), or within the simplifier (e.g. `for` loop in Maple). Average time:

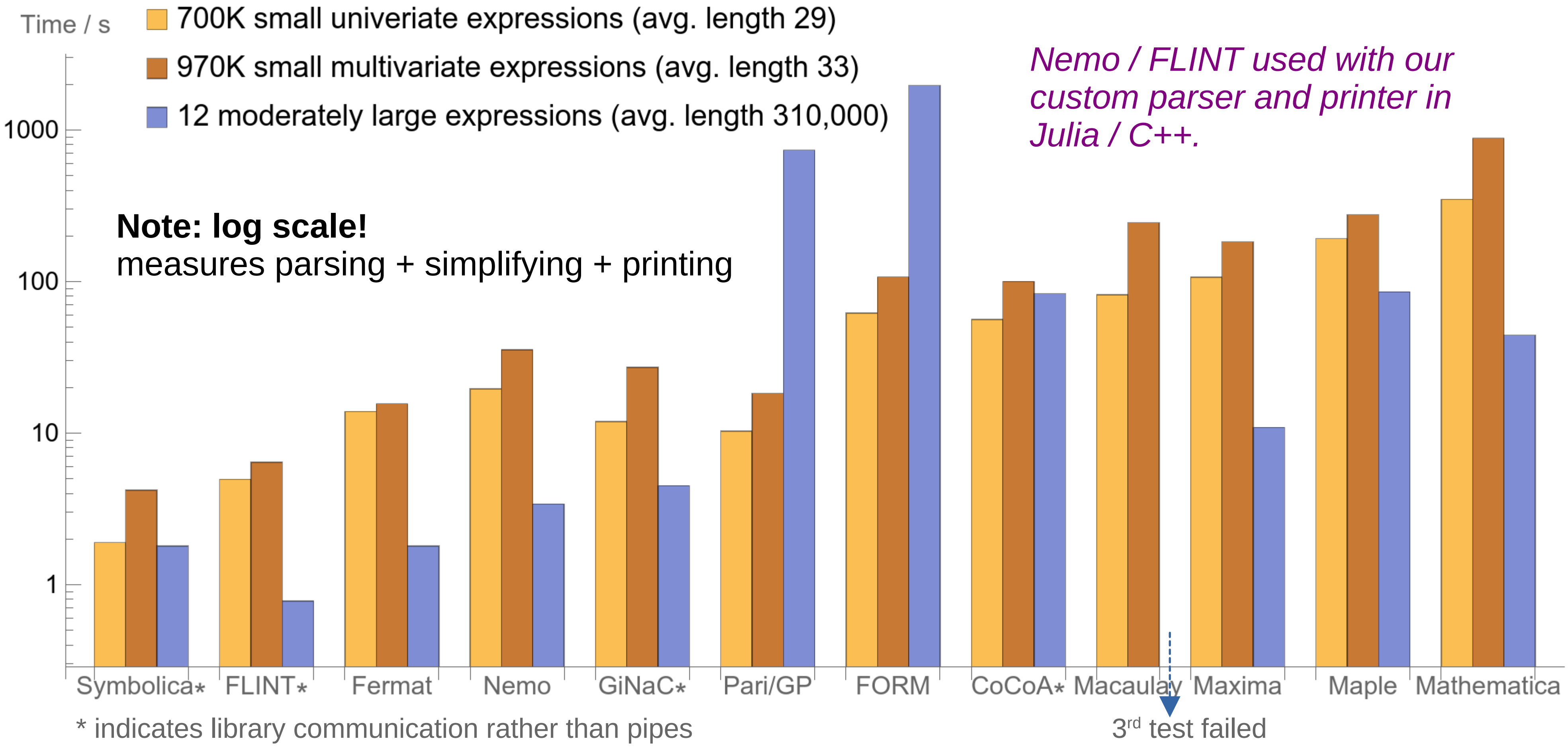
	via FUEL	within simplifier	Overhead
Fermat 5.17	14 μs	?	?
Maple 2022	180 μs	7 μs	×25
Mathematica 13.0	550 μs	40 μs	×13

- **Overhead** in parsing (string to expression) & printing (reverse).

What can slow down FIRE?

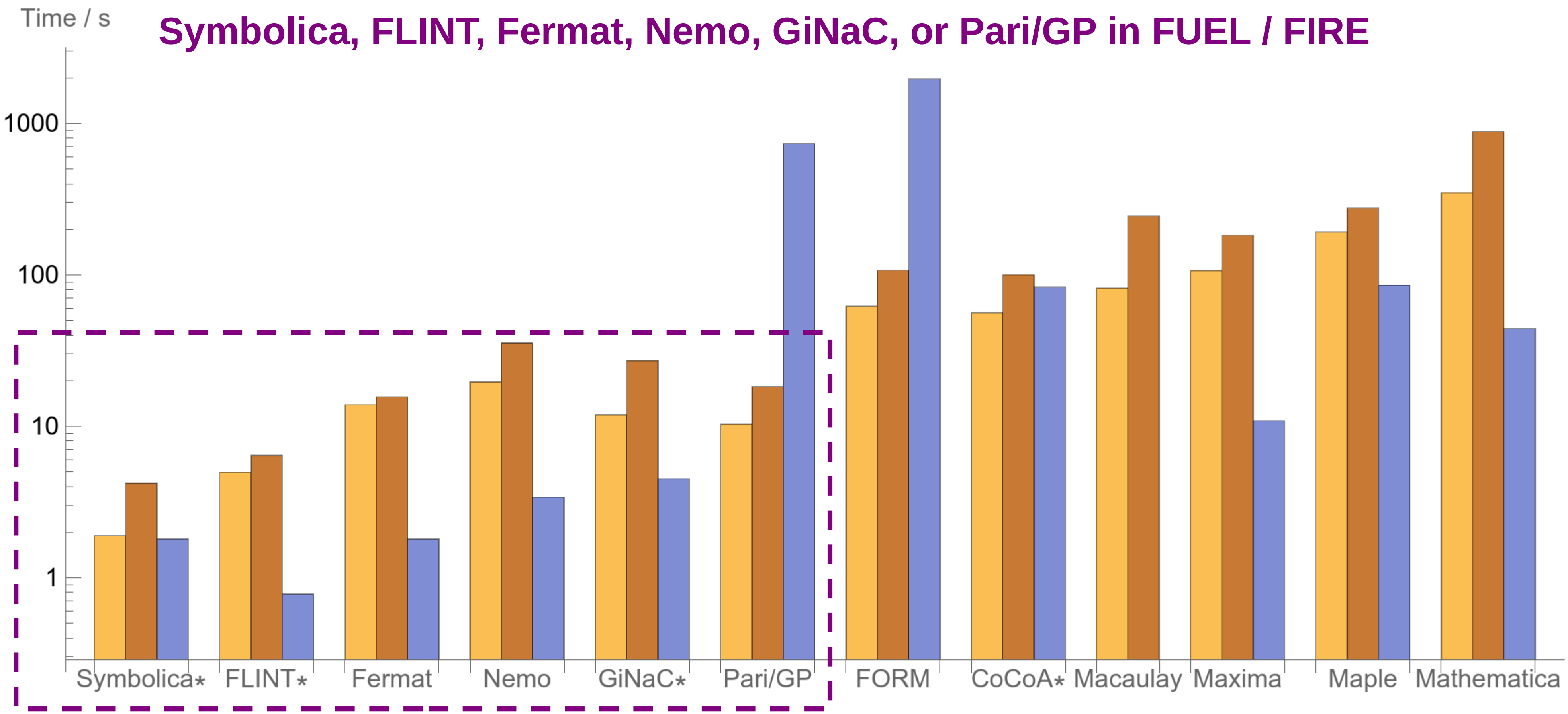
- **Parsing overhead.** e.g. Mathematica & Maple parse any statements in their languages. Dedicated parser for rational function expressions can be much faster. *(e.g. Dijkstra's shunting yard algorithm)*
- **Re-evaluation cost.** Simplified expression strings `(poly1)/(poly2)` re-inserted into further computations by string contcatenation, triggering *redundant polynomial GCD computations.*
- Example: Nemo CAS for Julia language, with a top-performing polynomial GCD engine (FLINT). Initial performance very poor. Achieved top performance after we implemented a custom parser in Julia, and a custom print format rat[poly1, poly2] to mark simplified expressions.
- Since initial paper, strategy reused in new backends: Symbolica, FLINT

Test: Small to moderately large expressions

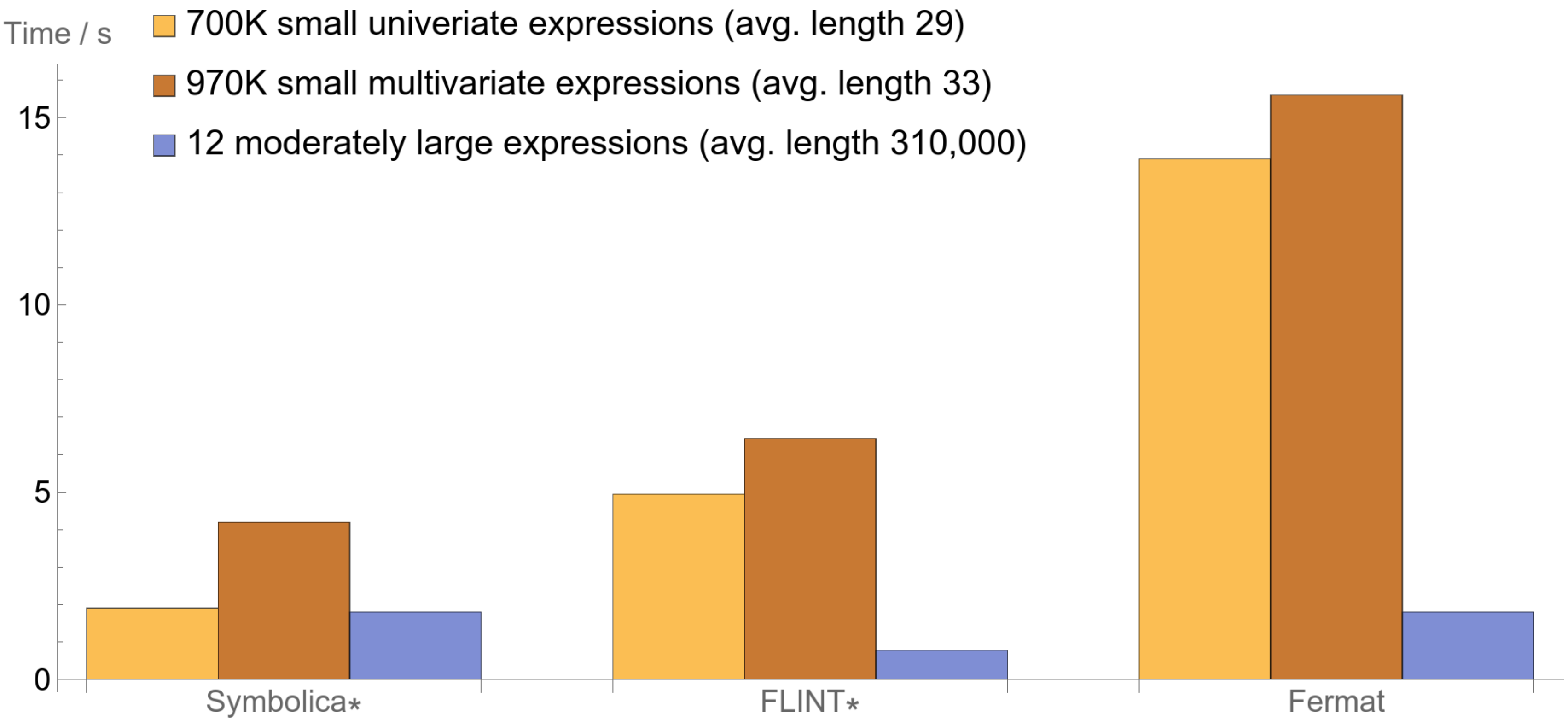


* indicates library communication rather than pipes

Best options for easy IBP problems



Linear scale plots for top options



Test: huge expression

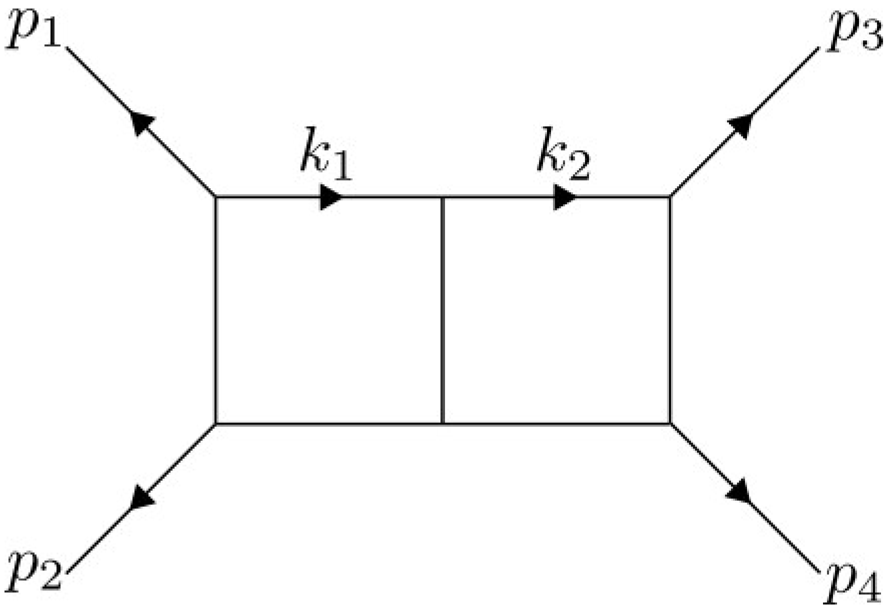
$$x = \frac{(a+b+c+d+f+g)^{14}+3}{(2a+b+c+d+f+g)^{14}+4} - \frac{(3a+b+c+d+f+g)^{14}+5}{(4a+b+c+d+f+g)^{14}+6}$$

- Parsing / printing overhead small compared with actual calculation

Simplifier	Time taken via FUEL (seconds)
FLINT	5.2
Symbolica	5.2
Nemo	6.9
Maple	7.9
Fermat	98.3
Maxima	112.8
Mathematica	169

FUEL in FIRE: easy IBP test

- Private version of FIRE using FUEL, to reduce following integral



$$\times\ (k_2+p_1)^2(k_1-p_3)^2$$

FUEL in FIRE: easy IBP test

Simplifier	Total Time	Substitution time	Memory usage (FIRE + simplifier)
Symbolica (lib)	14.6	1.1	15.6
Fermat	19.8	1.8	23.0
Pari / GP	21.6	2.3	14.1
Nemo	32.8	2.4	431.7
GiNaC (lib)	27.6	6.0	14.6
CoCoA (lib)	73.2	4.7	16.0
FORM	78.5	5.8	14.4
Maxima	131.2	6.7	955.8
Macaulay	152.5	12.1	350.9
Maple	177.6	6.5	105.1
Wolfram Mathematica	581.0	22.2	146.1

Preliminary result for FLINT: slower than Symbolica, faster than Fermat

FUEL in FIRE: easy IBP test

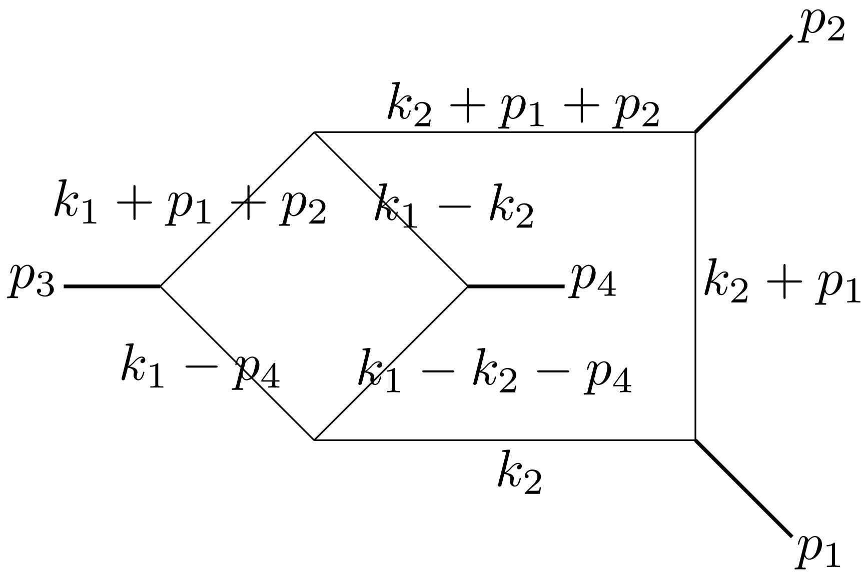
Simplifier	Total Time	Substitution time	Memory usage (FIRE + simplifier)
Symbolica (lib)	14.6	1.1	15.6
Fermat	19.8	1.8	23.0
Pari / GP	21.6	2.3	14.1
Nemo	32.8	2.4	431.7
GiNaC (lib)	27.6	6.0	14.6
CoCoA (lib)	73.2	4.7	16.0
FORM	78.5	5.8	14.4
Maxima	131.2	6.7	955.8
Macaulay	152.5	12.1	350.9
Maple	177.6	6.5	105.1
Wolfram Mathematica	581.0	22.2	146.1

backward substitution more demanding than forward elimination

Preliminary result for FLINT: slower than Symbolica, faster than Fermat

FUEL in FIRE: harder IBP test

- Private version of FIRE using FUEL, to reduce following integrals, with rank ≤ 2 numerator, from massive form factors in *N*=4 SYM on Coloumb branch [A.V. Belitsky, L.V. Bork, V.A. Smirnov, in progress]



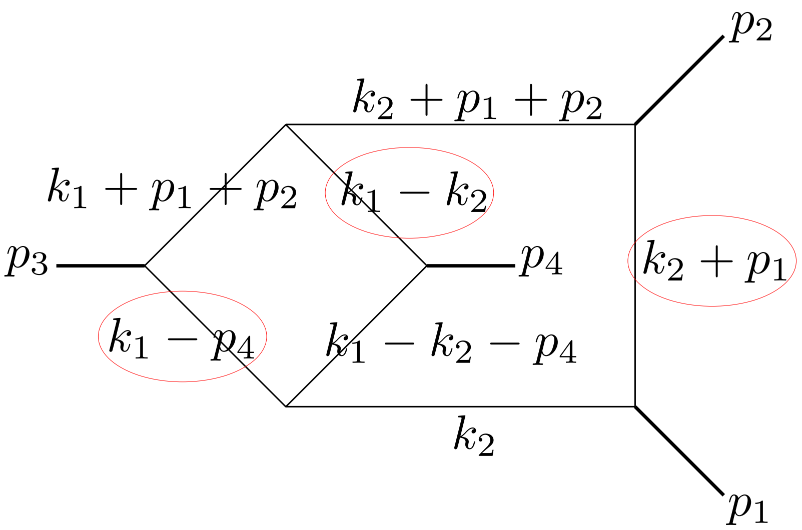
$$p_1^2 = p_2^2 = p_3^2 = -m^2,$$
$$(p_1 + p_2)^2 = -u,$$
$$(p_2 + p_3)^2 = -v,$$
$$(p_3 + p_1)^2 = -w$$

5 variables including spacetime dimension d

FUEL in FIRE: harder IBP test

- Time to obtain coefficient of bottom-level sunrise master integral, setting other master integrals to zero. *More than 10 times speedup!*

(Backward substitution dominates)

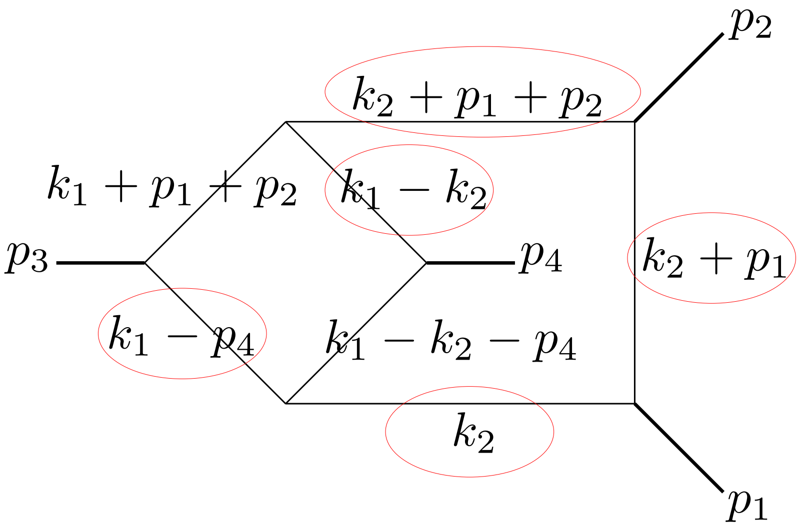


Simplifier	Time (seconds)
Symbolica	7,700
FLINT	8,400
Nemo	11,300
Fermat	104,000

FLINT / Nemo used with our custom parser and printer.

FUEL in FIRE: harder IBP test

- Time to obtain coefficient of 5-propagator box-bubble master integral, setting other master integrals to zero.



Simplifier	Time (seconds)
FLINT	265
Symbolica	343
Fermat	1820

FLINT wins this one. ~7 times speedup w.r.t. Fermat

Conclusions

- Investigated efficient use of computer algebra systems in IBP.
- Historically, external simplifier (Fermat) used as black box, “string in, string out”, by C++ IBP programs FIRE, Kira, Reduze.

  - Good performance requires fast polynoimal GCD + fast parsing / printing
- There are faster polynomial GCD engines than Fermat – **Their use in FIRE is made practical** by writing fast parsers for these engines.

  Possiblity: skip string intermediary? (obstructions: database, compression, inter-process exchange of expressions)
- ***Vast speedup*** for multi-scale IBP w/ Symbolica, FLINT, Nemo.
- To do: test more demanding problems; compare with finite field

Acknowledgments

- We thank the authors of FLINT, Nemo and FORM for help with our questions about the software in mailing lists and/or private communications.
- We especially thank the author of Symbolica, Ben Ruijl, for tirelessly answering our questions and customizing the software to integrate with our library.
- The work of Alexander Smirnov was supported by the Russian Science Foundation under the agreement No. 21-71-30003.
- M.Z.'s work is supported in part by the U.K. Royal Society through Grant URF\R1\20109.