2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

doi 10.26089/NumMet.v27r326

Factorization of denominators as a 'fuel' for Feynman integral reduction

Alexander V. Smirnov
Research Computing Center, Moscow State University, Moscow, Russia
Moscow Center for Fundamental and Applied Mathematics,
Moscow State University, Moscow, Russia
ORCID: 0000-0002-7779-6735, e-mail: asmirnov@srcc.msu.ru

Vladislav A. Fokin
Lomonosov Moscow State University,
Faculty of Computational Mathematics and Cybernetics, Moscow, Russia
ORCID: 0009-0007-3800-8701, e-mail: v.fokin2004@gmail.com

Egor Iu. Chuvashov
Lomonosov Moscow State University,
Faculty of Computational Mathematics and Cybernetics, Moscow, Russia
ORCID: 0009-0003-1157-0834, e-mail: wowonline.fit@mail.ru

Abstract: Rational-function simplification is a key bottleneck in integration-by-parts (IBP) reduction of Feynman integrals. We study denominator factorization patterns appearing in IBP coefficients and develop practical algorithms for extracting and exploiting factorized denominator structure within the `FUEL` interface. The resulting workflow reduces reconstruction cost and improves robustness of large-scale reductions.

Keywords: Feynman integrals, IBP reduction, FUEL, denominator factorization.

Acknowledgements: The work of A. V. Smirnov was supported by the Ministry of Education and Science of the Russian Federation as part of the program of the Moscow Center for Fundamental and Applied Mathematics under Agreement No. 075–15–2025–345. We are grateful to Mao Zeng for assisting with examples and conducting the performance measurements.

For citation: A. V. Smirnov, V. A. Fokin, and E. Iu. Chuvashov, "Factorization of denominators as a 'fuel' for Feynman integral reduction," Numerical Methods and Programming. **27** (3), 406–416 (2026). doi 10.26089/NumMet.v27r326.

© Alexander V. Smirnov, Vladislav A. Fokin, Egor Iu. Chuvashov

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

Факторизация знаменателей как метод ускорения редукции интегралов Фейнмана

А. В. Смирнов
Московский государственный университет имени М. В. Ломоносова,
Научно-исследовательский вычислительный центр,
Москва, Российская Федерация
Московский государственный университет имени М. В. Ломоносова,
Московский Центр фундаментальной и прикладной математики,
Москва, Российская Федерация
ORCID: 0000-0002-7779-6735, e-mail: asmirnov@srcc.msu.ru

В. А. Фокин
Московский государственный университет имени М. В. Ломоносова,
факультет вычислительной математики и кибернетики,
Москва, Российская Федерация
ORCID: 0009-0007-3800-8701, e-mail: v.fokin2004@gmail.com

Е. Ю. Чувашов
Московский государственный университет имени М. В. Ломоносова,
факультет вычислительной математики и кибернетики,
Москва, Российская Федерация
ORCID: 0009-0003-1157-0834, e-mail: wowonline.fit@mail.ru

Аннотация: Упрощение рациональных функций является основным процессом, ограничивающим производительность редукции интегралов Фейнмана методом интегрирования по частям (IBP). В настоящей работе применяется подход, связанный с факторизацией знаменателей, возникающих в коэффициентах IBP-редукции, и предлагается алгоритм для эффективного использования факторизованной структуры в интерфейсе `FUEL`. Разработанный подход позволяет снизить вычислительные затраты и увеличить производительность.

Ключевые слова: интегралы Фейнмана, IBP редукция, FUEL, факторизация знаменателей.

Благодарности: Работа А. В. Смирнова поддержана Министерством просвещения и науки Российской Федерации в рамках программы Московского центра фундаментальной и прикладной математики по соглашению № 075–15–2025–345. Авторы выражают благодарность Мао Цзэну за помощь в подборе примеров и в проведении измерений производительности.

Для цитирования: Смирнов А.В., Фокин В.А., Чувашов Е.Ю. Факторизация знаменателей как метод ускорения редукции интегралов Фейнмана // Вычислительные методы и программирование. 2026. **27**, № 3. 406–416. doi 10.26089/NumMet.v27r326.

---

1. Introduction. The evaluation of Feynman integrals is a central task in perturbative quantum field theory calculations. It is based on the integration-by-parts (IBP) reduction approach [1, 2], which makes it possible to represent all integrals belonging to a given family (defined by a particular set of propagators) in terms of a finite set of simpler integrals known as master integrals.

Reduction generally means combining IBP identities in a nontrivial manner. A systematic procedure for this is the so-called Laporta algorithm [3]. Several public implementations of this algorithm exist, including `AIR` [4], `Reduze` [5, 6], `LiteRed` [7–9], `FIRE` [10–15], and `Kira` [16–18].

There are two competitive approaches to Feynman integral reduction. The first one is the classical approach, wherein the system of equations is solved directly. The second one is the modular approach, wherein reduction is performed multiple times for fixed values of kinematic invariants and then, a reconstruction of an analytic function is carried out [13, 17–35].

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

In this paper, however, we will focus mainly on the classical approach, in which the reduction procedure technically represents the solution of a large sparse linear system with polynomial coefficients. Nevertheless, our method is only applicable within the modular approach to speed up the reconstruction stage.

One should pay a close attention to structures of expressions appearing in denominators of arbitrary Feynman integrals when the latter are reduced to master integrals. The denominators can be considered as poles, which for Feynman integrals can exist only for specific values of kinematic invariants. In particular, it is known (while not strictly proven) that one can always choose such a basis of master integrals that the denominators can be factorized (or split) into a product of a part depending on the spacetime dimension d and a part depending on other kinematic variables [36, 37].

Moreover, the denominators tend to be repeating and appear in powers, which leads to an idea that it might be useful to keep them factorized in order to speed up the reduction. A common objection is that the factorization is an algorithmically complicated operation and its carrying out might degrade performance. But as was initially suggested in the work [38] on `Symbolica`, the need to factorize might happen not very often, so that there is a chance to obtain performance gain. The `Symbolica` library has already had such an option for years that can be turned on by the `FUEL` interface. However, this fact is apparently not widely known or used mainly because `Symbolica` is a commercial package.

The aim of this paper is to implement denominator factorization using the `FLINT` library [39]. The main problem is that although `FLINT` has a class representing a factorized polynomial it has no arithmetical operations on such objects. Thus, one needs to implement a logic of working with such objects in `FUEL` [40]. Communication between `FUEL` and `FIRE` consists of sending strings to each other, and such a string should represent a polynomial. In case one needs to keep the factorized structure, `FIRE` should be able to store such structure after obtaining it from `FUEL` and then send it back, and the `FUEL` parser should be able to interpret the expression without joining the denominator factors. Eventually, the implementation of the discussed ideas leads to a creation of a rather complicated code.

In this paper, we describe realization stages of denominator factorization using `FLINT` in the `FUEL` interface and discuss how they can be used in `FIRE`. This also leads to a need to describe the proposed decomposed format for storing `FLINT` expressions and its implementation. We also provide benchmarks confirming the efficiency of such an approach for problems with multiple kinematic invariants.

The remainder of this paper is organized as follows. In Section 2, we describe our implementation of the factorization approach within the `FUEL` interface using `FLINT`. In Section 3, we provide instructions how to use the new approach with `FIRE`. Section 4 is devoted to a study of performance benchmarks and measurements.

2. Implementation in `FUEL`. First of all, one needs to remind how `FIRE` works with the `FUEL` interface and how other IBP reduction programs might use `FUEL`. The main `FUEL` evaluation function receives a string containing an expression as an input and returns also a string with a simplified expression. Depending on options, this expression might have a human-readable setup (in any case, it is necessary before saving the final result) or take any other format. The only severe restriction imposed by `FIRE` is that a pure zero result must always be returned as "0". `FIRE` checks whether the expression is "0" and if it is not identically zero, this expression is considered as a non-zero rational function that can appear in a denominator. Now `FIRE` works with these expressions, adding them to each other, multiplying, dividing, subtracting. But `FIRE` does not have any built-in algebraic system, so the operations are pure string expressions. For example, after receiving expressions `expr1` and `expr2` from `FUEL`, the package `FIRE` can "add them to each other", resulting in `(expr1 + expr2)`. Afterwards, such a combined expression can be sent back to the `FUEL` parser, which interprets and simplifies it properly. Thus, `FUEL` is free to use any optimized output for storing expressions, unless `FIRE` directly requests the result in a human-readable form with a single restriction to return a pure zero as "0".

2.1. Decomposed output without factorization. To support the storage of factorized expressions and to speed up parsing, we introduced a `decomposed output` for polynomials. This format was first implemented in an unpublished coursework project of E. Iu. Chuvashov.

In the standard `FIRE`–`FUEL` workflow rational functions are represented as fractions of multivariate polynomials

$$f(x_1, \ldots, x_n) \;=\; \frac{p(x_1, \ldots, x_n)}{q(x_1, \ldots, x_n)}\,,$$

where the polynomials are stored internally as sums of monomials. To make parsing and serialization efficient, `FUEL` does not print full symbolic expressions such as $(x + y^2 - 5z^{10})/(2x + z^2)$, but instead uses a `decomposed output` format that exposes the monomial structure explicitly.

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

Each monomial in n variables

$$m(x_1,\dots,x_n) = c\,x_1^{k_1}x_2^{k_2}\cdots x_n^{k_n}, \qquad c\in\mathbb{Z},\ k_i\in\mathbb{N}_0=\mathbb{N}\cup\{0\}$$

is represented in a serialized compact record containing the coefficient followed by all exponents:

$$m \longmapsto \langle c;\, k_1, k_2, \dots, k_n\rangle,$$

where in the actual serialized output the entries are separated by dots and typeset below in `typewriter` font. The ordering and number of variables $(x_1,\dots,x_n)$ are fixed by the `FIRE` problem definition, so the variable names themselves do not have to be printed. Zero exponents $k_i = 0$ are kept explicitly in the record so that the length of the exponent list is constant.

A polynomial

$$p(x_1,\dots,x_n) = \sum_j m_j(x_1,\dots,x_n)$$

is then represented as a concatenation of monomial records

$$p \longmapsto \langle c_1;\, k_{1,1},\dots,k_{1,n}\rangle\ \langle c_2;\, k_{2,1},\dots,k_{2,n}\rangle\ \cdots$$

with the same dot-separated encoding for each individual monomial in the actual output as above. A rational function p/q is printed as a pair of such sequences, enclosed in an additional layer of angle brackets that separates the numerator and denominator. For example, the rational function

$$\frac{p(x,y,z)}{q(x,y,z)} \;=\; \frac{x+y^2-5z^{10}}{2x+z^2}$$

has the schematic decomposed output

```
<{⟨1.1.0.0⟩ ⟨1.0.2.0⟩ ⟨-5.0.0.10⟩} . {⟨2.1.0.0⟩ ⟨1.0.0.2⟩}>
```

The outermost pair of brackets delimits the fraction, while the two inner bracketed blocks contain the monomials of the numerator and denominator, respectively. In this representation `FUEL` can build and modify polynomials and rational functions incrementally, adding monomials "on the fly" without invoking the slower general-purpose parser for human-readable algebraic expressions.

2.2. Decomposed output with factorized denominators. For factorized denominators, we extend the decomposed output format so that the internal `FLINT` representation `fmpz_mpoly_factor_struct` can be reconstructed directly in a single pass over the serialized string. Internally, a factorized polynomial

$$D(x_1,\dots,x_n) \;=\; c\prod_{i=1}^{r} p_i(x_1,\dots,x_n)^{e_i}, \qquad c\in\mathbb{Q},\ e_i\in\mathbb{N}_0$$

is stored as an array of simple polynomial factors p_i together with their exponents e_i and an overall rational constant c.

In the factorized decomposed output, the numerator is still printed as a sequence of monomials in the basic format described above. The denominator, however, is encoded in a nested form that contains two flags, a rational constant, and the list of factors:

$$\Big\langle \mathrm{flag}_1\ \mathrm{flag}_2 \Big\langle c_{\mathrm{num}}.c_{\mathrm{den}} \Big\langle e_1\langle p_1^{\mathrm{dec}}\rangle \cdots e_r\langle p_r^{\mathrm{dec}}\rangle \Big\rangle \Big\rangle \Big\rangle,$$

where p_i^{dec} denotes the polynomial p_i written in decomposed form. Here

- flag_1 and flag_2 are two Boolean flags indicating whether factorization should be applied and whether the denominator is already stored in factorized form;
- `c_num.c_den` is the rational constant in front of the product of factors written as a numerator–denominator pair;
- each factor is written in the form $e_i\langle p_i^{\mathrm{dec}}\rangle$, where e_i is the exponent and p_i^{dec} is the corresponding polynomial factor in decomposed form

$$p_i(x_1,\dots,x_n) \longmapsto \langle c_{i,1};\, k_{i,1,1},\dots,k_{i,1,n}\rangle\ \langle c_{i,2};\, k_{i,2,1},\dots,k_{i,2,n}\rangle\ \cdots$$

with the same dot-separated encoding for each individual monomial in the actual output as above.

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

Schematically, a rational function with a factorized denominator is serialized as

$$\Big\langle \underbrace{\langle \text{monomials of numerator} \rangle}_{\text{numerator}} \ . \ \underbrace{\langle \text{flag}_1\, \text{flag}_2 \langle c_{\text{num}}.c_{\text{den}} \langle e_1 \langle p_1^{\text{dec}} \rangle \cdots e_r \langle p_r^{\text{dec}} \rangle \rangle \rangle \rangle}_{\text{factorized denominator}} \Big\rangle.$$

During parsing, `FUEL` reads the numerator monomials as before, then interprets the two flags and the constant pair, and finally reconstructs the factors p_i with their exponents e_i to populate an `fmpz_mpoly_factor_struct`. This allows arithmetic operations to exploit the factorized structure directly (e.g. cancelling common factors or updating exponents) without repeatedly multiplying out large denominator polynomials.

For example, let us consider a fraction

$$\frac{N(x,y,z)}{D(x,y,z)} \ = \ \frac{x + y^2 - 5z^{10}}{\frac{7}{5}\,(x-y)^2(2x+z^2)}.$$

The numerator is printed in the same way as in the non-factorized case:

$$N(x,y,z) \ \longmapsto \ \texttt{⟨1.1.0.0⟩ ⟨1.0.2.0⟩ ⟨-5.0.0.10⟩}$$

For the denominator, the constant prefactor is written as

$$\frac{7}{5} \longmapsto \texttt{7.5}$$

and the two polynomial factors are written as

$$x - y \longmapsto \texttt{⟨1.1.0.0⟩ ⟨-1.0.1.0⟩}, \qquad 2x + z^2 \longmapsto \texttt{⟨2.1.0.0⟩ ⟨1.0.0.2⟩}.$$

Thus, in the actual syntax used in the code, the full serialized expression is

```
<{⟨1.1.0.0⟩ ⟨1.0.2.0⟩ ⟨-5.0.0.10⟩} . {11{7.5{2{⟨1.1.0.0⟩ ⟨-1.0.1.0⟩} 1{⟨2.1.0.0⟩ ⟨1.0.0.2⟩}}}}>
```

Here two digits `11` are Boolean flags written sequentially: the first flag is the value of `factorize_denominators`, and the second one indicates whether the denominator is already stored in a factorized form. The part `{7.5 {···}}` means that the overall constant is $7/5$, and after that the list of factors follows.

In particular, the block

```
2{⟨1.1.0.0⟩ ⟨-1.0.1.0⟩}
```

should be read in the following way: the first number 2 is the exponent of the factor, and the part inside brackets is the decomposed form of the polynomial

$$\texttt{⟨1.1.0.0⟩ ⟨-1.0.1.0⟩} \longmapsto x - y.$$

As a result, this whole block represents the factor $(x-y)^2$. In the same way, the denotation

```
1{⟨2.1.0.0⟩ ⟨1.0.0.2⟩}
```

represents the factor $(2x + z^2)^1$. Thus, each factor in the denominator is written as `e_i`$\langle p_i^{\text{dec}} \rangle$: first the exponent e_i, then the polynomial factor itself in the usual decomposed monomial format.

2.3. Algorithms working with factorized denominators. Since `FLINT` does not support operations on expressions with factorized denominators, we had to implement our own routines for handling such functions. Now let us suppose, there are two expressions in a decomposed form with factorized denominators and we need to perform an algebraic operation on them. To be more exact, let us take expressions

$$\frac{h}{g} = \frac{h}{c\, g_1^{k_1} g_2^{k_2} \cdots g_m^{k_m}} \quad \text{and} \quad \frac{h'}{g'} = \frac{h'}{c'\, {g'_1}^{k'_1} {g'_2}^{k'_2} \cdots {g'_l}^{k'_l}}$$

The following operations should be considered.

§ 2.3.1. Multiplication. It is necessary to keep the expression in a proper format so that the product remains factorized and the numerator has no common factors with denominator. Also we need to avoid slow calls of factorization. First of all, it is necessary to reduce the numerator h with the factors of the denominator g'.

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

We try to reduce by each factor g_i' until its exponent allows, provided that g_i' is not present among the factors of its own denominator g. This reduces the number of divisions, which are computationally expensive. This is possible due to the fact that the input coefficients are already reduced, so the numerator h cannot be divided by the factors of its own denominator g. Similarly, we reduce the numerator h' by factors of the denominator g. After the reduction, one needs multiply the remaining parts of the numerators forming the numerator of the result. The denominator of the result includes all the remaining factors of both denominators after the reduction. Here we take advantage of factorization by simply adding factors to the result. If a factor appears in both denominators, we merely sum their exponents. The constants c and c' are multiplied, and the resulting constant $c'' = c''_{\text{num}}/c''_{\text{den}} = c \cdot c'$ is reduced so that $\gcd(c''_{\text{num}}, c''_{\text{den}}) = 1$. After that, we find the greatest common numerical constant k of all monomials of the resulting numerator, compute $\gcd(k, c''_{\text{num}})$ and simplify the fraction. This prevents the growth of numerical coefficients and denominator constants in subsequent computations.

§ 2.3.2. Addition and subtraction. Addition and subtraction are common operations differing only by the sign of the second expression. The basic formula for addition is $(h \cdot g' + h' \cdot g)/(g \cdot g')$. First, we bring fractions to a common denominator by multiplying the numerators of each fraction by the missing factors from the opposite denominator. After that, we add or subtract the obtaining numerators forming an intermediate result. Similarly, we find the largest common numerical constant k of all monomials of the resulting numerator, compute $\gcd(k, c''_{\text{num}})$ of k and the constant of the resulting denominator c''_{num}, and simplify the fraction to keep numerical coefficients small. Next, we try to reduce the numerator by those factors of the resulting denominator that appeared in both original denominators with the same exponents. It makes no sense to try dividing by other factors because the input coefficients are already reduced. For example, one may consider the expression $\dfrac{h}{f \cdot g} + \dfrac{h'}{g} = \dfrac{h + f \cdot h'}{f \cdot g}$. Here h is not divisible by f, while the product $f \cdot h'$ is, which means the sum will not be divisible by f.

§ 2.3.3. Division. Division is the only operation that can not be performed without calling factorization. Fortunately, it is needed much less often than other operations during reduction. We factorize the numerator h' and then use the identity $\dfrac{h}{g} \Big/ \dfrac{h'}{g'} = \dfrac{h}{g} \cdot \dfrac{g'}{h'}$, similarly to the case of factorized multiplication. When reducing g' against the factors of g, we take advantage of the factorized form of g' matching the coinciding factors. We do not call expensive polynomial divisions and instead simply reduce the exponents of common factors.

§ 2.3.4. Exponentiation. Raising a coefficient to an integer power e is relatively rare. If $e > 0$, we raise the numerator and the denominator constant to the power e, and multiply the exponents of the denominator factors by e. If $e < 0$, we use $\left(\dfrac{h}{g}\right)^e = \left(\dfrac{g}{h}\right)^{-e}$. In this case, one needs to factorize the numerator h and expand the denominator g (i.e., convert it to a dense polynomial). Otherwise, it is similar to the case of a positive power.

3. Usage in `FIRE`. In this section we provide instructions on how to use the described improvements within the `FIRE` package. It is assumed that `FIRE7` is already installed. To get the possibility to use factorized denominators with `FUEL`, one has to switch to a release starting from version 7.1. Then one runs `make fuel` to get the latest submodule update, followed by `make clean` and `make` to rebuild the code. Now `FUEL` is ready to accept the new options.

The syntax is to use the `calc_options` command of `FIRE` with

```
--calc_options decomposed_output, factorize_denominators
```

One can also install `Symbolica`, obtain a proper license and put it into `/.symbolica/main_license` file to be able to use it with `-calc symbolica`. `Symbolica` evaluator also accepts `factorize_denominators`, while `decomposed_output` is not required here and will have no effect.

4. Benchmarks.

4.1. Benchmarks on physically motivated examples. In this section we present a number of benchmarks demonstrating the performance of the new approach. We consider a number of physically motivated examples and compare the code in the following modes:

- Default with `-calc flint`;
- Decomposed output mode with `-calc flint`;

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

- Factorized mode (together with decomposed output) with `-calc flint`;
- Using of `-calc symbolica` as a simplification library;
- Factorized mode with `-calc symbolica`.

The results below do not reveal a clear winner in the competition between our approach with `FLINT` and `Symbolica`, but clearly demonstrate the efficiency of our approach compared with pure `FLINT` runs.

§ 4.1.1. Two-loop massive nonplanar double box. Our first example is a 2-loop diagram shown in Fig. 1 with 5 variables that we already used in [40]. The legs p_1, p_2 and p_3 have mass m, while the leg p_4 is massless. We perform IBP reduction of the three tensor integrals with numerators $(k_1+p_1)^2$, $(k_2-p_4)^2$ and $(k_1+p_1)^2(k_2-p_4)^2$, respectively.

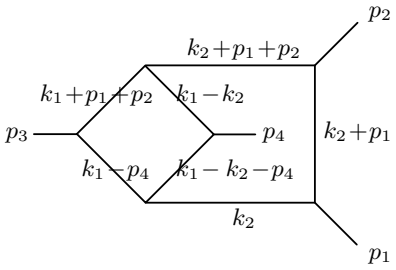

Figure 1. The nonplanar two-loop double box diagram with three off-shell legs

We run a number of tests restricting the possible level of non-zero master integrals (by setting a part of them to zero) and thus modifiying the complexity of the reduction. The results, obtained on a 4-core AMD Ryzen 7 3750H laptop, are summarized in Table 1.

In this example `Symbolica` in factorized mode becomes faster when a larger set of master integrals is used. However, in one of our following examples the number of masters is even larger, and the results are opposite. The notation "RAM warning" in Table 1 means that 24 GB on the laptop was insufficient.

Table 1. Reduction times (in seconds) for the two-loop nonplanar double box

Level of masters	**7**	**6–7**	**5–7**	**4–7**
`FLINT`	22	896	13131	RAM warning
`FLINT` + decomposed	14	726	11769	RAM warning
`FLINT` + decomposed + factorized	10	396	4809	16361
`Symbolica`	20	1151	13725	RAM warning
`Symbolica` + factorized	23	890	4753	10643

§ 4.1.2. Nonplanar double pentagon diagram. Our second example is the nonplanar double pentagon diagram shown in Fig. 2, one of the 5-variable diagrams we used in our `FIRE` paper [15].

To speed up the benchmarking, we restricted the reduction to the top sector and ran a number of tests reducing an integral with different powers of irreducible denominators. The reduction was run on a cluster node using AMD EPYC 74F3 24-core processors with 16 cores reserved for the job.

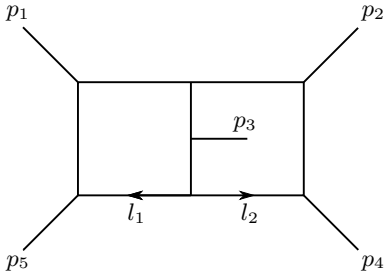

Figure 2. The nonplanar double pentagon diagram. The list of eight propagator denominators and three irreducible scalar products are: l_1^2, $(l_1-p_5)^2$, $(l_1-p_5-p_1)^2$, $(l_2-p_4-p_2)^2$, $(l_2-p_4)^2$, l_2^2, $(l_1+l_2)^2$, $(l_1+l_2+p_3)^2$, $(l_2+p_5)^2$, $(l_1+p_4)^2$ and $(l_1+p_4+p_2)^2$

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

In this test, `FLINT` with factorization of denominators showed a better performance than `Symbolica`. The results are presented in Table 2.

Table 2. Reduction times (in seconds) for the nonplanar double pentagon diagram

Numerators power	**1**	**2**	**3**	**4**
`FLINT`	2412	4271	11313	30668
`FLINT` + decomposed	1575	2782	7891	23321
`FLINT` + decomposed + factorized	900	1498	3915	12132
`Symbolica`	3390	5678	15066	39891
`Symbolica` + factorized	1722	2645	7308	20079

§ 4.1.3. Three-loop window. We took the third example from our `FIRE7` paper (section 5.3), namely a 3-loop integral family with massive internal lines. The kinematics corresponds to the exact forward scattering with massless incoming momenta, i.e. $q_1 + q_2 \to q_1 + q_2$. The diagram is shown in Fig. 3. In this benchmark, we reduce only sample integrals, where the 4th propagator with $((l_3 - q_2)^2 - m^2)$ is raised to the second power, and the total power of numerators varies.

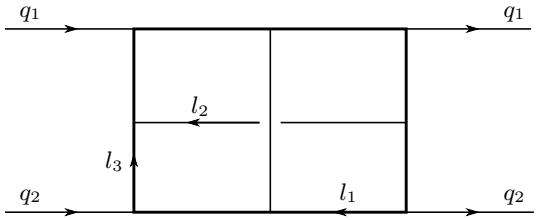

Figure 3. The three-loop forward-scattering diagram with massive internal lines and with $q_1^2 = q_2^2 = 0$, $(q_1 + q_2)^2 = s$. The list of 10 propagators are: $l_3^2 - m^2$, $(l_2 + l_3)^2 - m^2$, $l_1^2 - m^2$, $(l_3 - q_2)^2 - m^2$, $(l_2 + l_3 + q_1)^2 - m^2$, $(l_1 + q_2)^2 - m^2$, $(l_1 + l_2 + q_1 + q_2)^2 - m^2$, $(l_1 + l_2 + q_2)^2 - m^2$, l_2^2 and $(l_1 - l_3 + q_2)^2$. The two irreducible numerators are $(l_1 + l_2 + l_3)^2$ and $(l_1 - q_1)^2$

We obtained the following results on the same 4-core AMD Ryzen 7 3750H laptop, presented in Table 3.

One can see that `FLINT` works approximately at the same speed as `Symbolica` with the factorized approach. The performance gain of the factorization is modest, but the gain in saving RAM is appreciable. Indeed, there was not enough RAM to run without factorization at our disposal. It is also important that the decomposed output mode is essential compared with pure `FLINT`.

Table 3. Reduction times (in seconds) for the three-loop forward-scattering diagram

Level of numerators	**0**	**1**	**2**	**3**
`FLINT`	1598	5133	6884	RAM warning
`FLINT` + decomposed	1045	3298	4681	RAM warning
`FLINT` + decomposed + factorized	1030	3290	4352	16826
`Symbolica`	1131	3325	5016	RAM warning
`Symbolica` + factorized	1002	3183	4422	15886

4.2. Internal code measurements. Our goal was also to verify that our code is properly optimized, so we performed some internal performance measurements. For a big enough example we chose the two-loop massive nonplanar double box from section 4.1.1 with master integrals of levels 5–7. We ran measurements with `google perf`. Our goal was to measure the time spent in the `fuel::base::evaluate` function called by `FIRE` for expression simplification. The time usage based on collected stack data is as follows:

- 34.30% of time is spent on addition calls;
- 56.19% of time is spent on multiplication calls;
- 1.08% of time is spent on division calls (where factorization happens);
- The remaining part of time is spent on string manipulations, memory operations and expression output.

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

We also tested this on examples of different size and clearly observed that the division and factorization become negligible for sufficiently large expressions.

As for the large parts for addition and multiplication, we demonstrated above how our algorithms perform them. The measurements on this example also showed that the overhead costs are small.

The addition can be split as follows:

- 31.07% of time is spent on `FLINT` multiplication calls;
- 45.52% of time is spent on `FLINT` division (or gcd) calls;
- 6.34% of time is spent on `FLINT` addition calls.

The multiplication can be split into the following parts:

- 75.28% of time is spent on `FLINT` multiplication calls;
- 18.99% of time is spent on `FLINT` division (or gcd) calls.

This measurement clearly shows that the overhead costs are quite small and that the implementation mostly relies on our algorithms (which we believe to be effective) and on the addition, multiplication and division routines inside the `FLINT` library, which are well-tested for years. Thus, no significant performance improvement can be expected with the current approach.

5. Conclusion. In this work we proposed to exploit the factorized structure of denominators that naturally appears in integration-by-parts reductions. We introduced a decomposed output format for polynomials and rational functions and extended it to support denominators stored as products of simple polynomials raised to integer powers, together with a corresponding serialization and parsing scheme. We implemented arithmetic operations that use the factorized form explicitly, avoiding unnecessary polynomial multiplications and enabling systematic cancellation of common factors already at the level of the serialized data. The resulting functionality is integrated into `FUEL` and can be used transparently from `FIRE` for problems with many kinematic invariants. To our knowledge, the only other implementation of the factorization of denominators in the IBP reduction workflow is the `Symbolica` library [38], which is a commercial package. Further work will focus on a more extensive performance study on realistic multi-scale IBP reduction and on measurements of performance gain in cases with more factors in the denominators and at higher integral indices.

References

1. K. G. Chetyrkin and F. V. Tkachov, "Integration by parts: The algorithm to calculate β-functions in 4 loops," Nucl. Phys. B **192** (1), 159–204 (1981). doi 10.1016/0550-3213(81)90199-1.
2. F. V. Tkachov, "A theorem on analytical calculability of 4-loop renormalization group functions," Phys. Lett. B **100** (1), 65–68 (1981). doi 10.1016/0370-2693(81)90288-4.
3. S. Laporta, "High-precision calculation of multiloop Feynman integrals by difference equations," Int. J. Mod. Phys. A **15** (32), 5087–5159 (2000). doi 10.1142/S0217751X00002159.
4. C. Anastasiou and A. Lazopoulos, "Automatic integral reduction for higher order perturbative calculations," JHEP **2004** (07), Article Number 046 (2004). doi 10.1088/1126-6708/2004/07/046.
5. C. Studerus, "Reduze — Feynman Integral Reduction in C++," Comput. Phys. Commun. **181** (7), 1293–1300 (2010). doi 10.1016/j.cpc.2010.03.012.
6. A. von Manteuffel and C. Studerus, "Reduze 2 — Distributed Feynman Integral Reduction," arXiv:1201.4330 [hep-ph] (2012). doi 10.48550/arXiv.1201.4330.
7. R. N. Lee, "Presenting LiteRed: a tool for the Loop InTEgrals REDuction," arXiv:1212.2685 [hep-ph] (2012). doi 10.48550/arXiv.1212.2685.
8. R. N. Lee, "LiteRed 1.4: a powerful tool for reduction of multiloop integrals," J. Phys. Conf. Ser. **523**, Article Number 012059 (2014). doi 10.1088/1742-6596/523/1/012059.
9. LiteRed2 Mathematica package. https://github.com/rnlg/LiteRed2/. Cited July 15, 2026.
10. A. V. Smirnov, "Algorithm FIRE — Feynman Integral REduction," JHEP **2008** (10), Article Number 107 (2008). doi 10.1088/1126-6708/2008/10/107.
11. A. V. Smirnov and V. A. Smirnov, "FIRE4, LiteRed and accompanying tools to solve integration by parts relations," Comput. Phys. Commun. **184** (12), 2820–2827 (2013). doi 10.1016/j.cpc.2013.06.016.
12. A. V. Smirnov, "FIRE5: A C++ implementation of Feynman Integral REduction," Comput. Phys. Commun. **189**, 182–191 (2015). doi 10.1016/j.cpc.2014.11.024.

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

13. A. V. Smirnov and F. S. Chukharev, "FIRE6: Feynman Integral REduction with modular arithmetic," Comput. Phys. Commun. **247**, Article Number 106877 (2020). doi 10.1016/j.cpc.2019.106877.
14. A. V. Smirnov and M. Zeng, "FIRE 6.5: Feynman integral reduction with new simplification library," Comput. Phys. Commun. **302**, Article Number 109261 (2024). doi 10.1016/j.cpc.2024.109261.
15. A. V. Smirnov and M. Zeng, "FIRE 7: Automatic reduction with modular approach," Comput. Phys. Commun. **325**, Article Number 110200 (2026). doi 10.1016/j.cpc.2026.110200.
16. P. Maierhöfer, J. Usovitsch, and P. Uwer, "Kira — A Feynman integral reduction program," Comput. Phys. Commun. **230**, 99–112 (2018). doi 10.1016/j.cpc.2018.04.012.
17. J. Klappert, F. Lange, P. Maierhöfer, and J. Usovitsch, "Integral reduction with Kira 2.0 and finite field methods," Comput. Phys. Commun. **266**, Article Number 108024 (2021). doi 10.1016/j.cpc.2021.108024.
18. F. Lange, J. Usovitsch, and Z. Wu, "Kira 3: Integral reduction with efficient seeding and optimized equation selection," Comput. Phys. Commun. **322**, Article Number 109999 (2026). doi 10.1016/j.cpc.2025.109999.
19. A. Díaz and E. Kaltofen, "FoxBox: a system for manipulating and solving algebraic differential equations," in *Proc. of ISSAC 1998* (1998).
20. A. von Manteuffel and R. M. Schabinger, "A novel approach to integration by parts reduction," Phys. Lett. B **744**, 101–104 (2015). doi 10.1016/j.physletb.2015.03.029.
21. T. Peraro, "Scattering amplitudes over finite fields and multivariate functional reconstruction," JHEP **2016** (12), Article Number 30 (2016). doi 10.1007/JHEP12(2016)030.
22. P. S. Wang, "A p-adic Algorithm for Univariate Partial Fractions," in *Proceedings of the Fourth ACM Symposium on Symbolic and Algebraic Computation (SYMSAC), USA, Snowbird Utah, August 5–7, 1981* (ACM, New York, 1981), pp. 212–217. doi 10.1145/800206.806398.
23. M. Monagan, "Maximal Quotient Rational Reconstruction: an Almost Optimal Algorithm for Rational Reconstruction," in *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC), Spain, Santander, July 4–7, 2004* (ACM, New York, 2004), pp. 243–249. doi 10.1145/1005285.1005321.
24. J. von zur Gathen and J. Gerhard, *Modern Computer Algebra* (Cambridge University Press, Cambridge, 2013). doi 10.1017/CBO9781139856065.
25. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (National Bureau of Standards, Washington, 1964).
26. R. Zippel, "Probabilistic Algorithms for Sparse Polynomials," in *Symbolic and Algebraic Computation. EUROSAM 1979.* Lecture Notes in Computer Science. Vol. 72. (Springer, Berlin, 1979), pp. 216–226. doi 10.1007/3-540-09519-5_73.
27. M. Ben-Or and P. Tiwari, "A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation," in *STOC'88: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, USA, Chicago Illinois, May 2–4, 1988* (ACM, New York, 1988), pp. 301–309. doi 10.1145/62212.62241.
28. R. Zippel, "Interpolating Polynomials from their Values," J. Symb. Comput. **9** (3), 375–403 (1990). doi 10.1016/S0747-7171(08)80018-1.
29. D. Grigoriev, M. Karpinski and M. F. Singer, "Computational Complexity of Sparse Rational Interpolation," SIAM Journal on Computing **23** 1, 1–11 (1994). doi 10.1137/S0097539791194069.
30. E. Kaltofen, W.-s. Lee, "Early termination in sparse polynomial interpolation," Journal of Symbolic Computation **36** 3, 365–400 (2003). doi 10.1016/S0747-7171(03)00088-9.
31. A. Cuyt, W.-s. Lee, "Sparse interpolation and rational approximation," in *Contemporary Mathematics: Modern Trends in Constructive Function Theory Held at the Vanderbilt University, Nashville, Tennessee, USA, May 26–30, 2014* (eds. D. P. Hardin, D. S. Lubinsky, B. Z. Simanek, AMS, Contemporary Mathematics, vol. 661, pp. 229–242). doi 10.1090/conm/661/13284. https://pubs.ams.org/ebooks/conm/661. Cited July 15, 2026.
32. T. Peraro, "FiniteFlow: multivariate functional reconstruction using finite fields and dataflow graphs," JHEP **2019** (7), Article Number 31 (2019). doi 10.1007/JHEP07(2019)031.
33. J. Klappert and F. Lange, "Reconstructing rational functions with FireFly," Comput. Phys. Commun. **247**, Article Number 106951 (2020). doi 10.1016/j.cpc.2019.106951.
34. A. V. Belitsky, A. V. Smirnov, and R. V. Yakovlev, "Balancing act: Multivariate rational reconstruction for IBP," Nucl. Phys. B **993**, Article Number 116253 (2023). doi 10.1016/j.nuclphysb.2023.116253.
35. A. V. Smirnov and M. Zeng, "Feynman integral reduction: balanced reconstruction of sparse rational functions and implementation on supercomputers in a co-design approach," Numerical Methods and Programming [Vychislitel'nye Metody i Programmirovanie] **25** (Special issue), 30–45 (2024). doi 10.26089/NumMet.2024s03.

2026, **27** (3), 406–416. doi 10.26089/NumMet.v27r326

36. A. V. Smirnov and V. A. Smirnov, "How to choose master integrals," Nucl. Phys. B **960**, Article Number 115213 (2020). doi 10.1016/j.nuclphysb.2020.115213.
37. J. Usovitsch, "Factorization of denominators in integration-by-parts reductions," arXiv:2002.08173v2 [hep-ph] (2020). https://arxiv.org/abs/2002.08173. Cited July 15, 2026.
38. Symbolica | Modern Computer Algebra. https://symbolica.io/. Cited July 15, 2026.
39. FLINT: Fast Library for Number Theory. https://flintlib.org. Cited July 15, 2026.
40. K. Mokrov, A. Smirnov, and M. Zeng, "Rational Function Simplification for Integration-by-Parts Reduction and Beyond," Numerical Methods and Programming [Vychislitel'nye Metody i Programmirovanie] **24** (4), 352–367 (2023). doi 10.26089/NumMet.v24r425.

Received *May 18, 2026*
Accepted *June 19, 2026*
Published *July 16, 2026*

Information about the authors

Alexander V. Smirnov — Dr. Sci., Professor; 1) Research Computing Center, Moscow State University, Leninskie Gory, 1, building 4, 119991, Moscow, Russia; 2) Moscow Center for Fundamental and Applied Mathematics, Moscow State University, Leninskie Gory, 1, 119991, Moscow, Russia.

Vladislav A. Fokin — Student; Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics, Leninskie Gory, 1, 119991, Moscow, Russia.

Egor Iu. Chuvashov — Bachelor; Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics, Leninskie Gory, 1, 119991, Moscow, Russia.