Recent developments on FORM

Takahiro Ueda[1] **and Jos Vermaseren**[2]

[1] Institut für Theoretische Teilchenphysik, Karlsruhe Institute of Technology (KIT), D-76128 Karlsruhe, Germany
[2] Nikhef Theory Group, Science Park 105, 1098 XG Amsterdam, The Netherlands

E-mail: takahiro.ueda@kit.edu, t68@nikhef.nl

Abstract. We report on recent developments on the open source computer algebra system FORM. Especially we focus on the code optimization implemented after the release of FORM version 4.0 in March 2012.

1. Introduction

FORM [1] is one of widely used programs for symbolic manipulation of huge formulas with an arbitrary number of terms appearing in theoretical calculations based on perturbative quantum field theories. Since March 2012 FORM version 4.0 has been released. The new features introduced in this version include:

- Polynomial factorization. FORM has a functionality of factorization of multivariate polynomials at three levels: function arguments, expressions and $-variables.
- Rational function arithmetic. FORM has a mode in which rational functions are considered as coefficients of terms in sorting. There are also functions which provide basic operations on polynomials: polynomial division, polynomial GCD and so on.

There are parallel versions of FORM, which automatically distribute the computation over many CPUs. One is TFORM [2], which is based on POSIX threads and works with multi-core processors. The other is PARFORM [3, 4] based on MPI, which can run on computer clusters with fast network connections. The above newly introduced features in the version 4.0 should also work with the parallel versions.

FORM is continuously being developed even after the release of the version 4.0, for some bug fixes and adding more features[1]. One of the recently added features, which will be considered as a part of the next release FORM 4.1, is code optimization.

After performing symbolic manipulation on formulas in an analytic way, one occasionally needs to export the result as a program code, e.g., in Fortran or C, for further numerical computations such as Monte Carlo integrations. If a lengthy formula will be evaluated millions of times, it is worth compactifying the expression such that the number of operations needed to evaluate it becomes as small as possible for fast computation, even if the optimization needs a

[1] One can access to the Git repository from `https://github.com/vermaseren/form`. For example, a bug in the disk-to-disk sorting, which may give a wrong result, was fixed on 23 July 2013. On the other hand, a new feature introduced on 13 August 2013 changed the store file format and executables compiled from the sources after this date cannot read old store files.

non-negligible time. FORM 4.1 will be equipped with the code optimization to reduce the cost of evaluations of multivariate polynomials, which includes an implementation of Monte Carlo tree search (MCTS) for finding a suitable multivariate Horner scheme [5, 6].

2. Code optimization in FORM

As the first example, let us consider the optimization of a polynomial F of three variables x, y and z:

```
Symbols x,y,z;
ExtraSymbols vector,v;
Local F = 6*y*z^2 + 3*y^3 - 3*x*z^2 + 6*x*y*z - 3*x^2*z + 6*x^2*y;
Format Fortran;
Format O1,stats=on;
Print;
.end
```

The special command for the optimization here is `Format O1,stats=on`. This enables the code optimization in printing the expression with reporting statistics. FORM has several optimization options and parameters. FORM also defines optimization levels `O0`, `O1`, `O2` and `O3` for user convenience, which are actually certain combinations of the options and parameters. Increasing the optimization level makes FORM try to give the output with fewer operations, at the cost of more time. The code optimization is internally applied to polynomials containing only numbers and symbols. Other objects are converted into symbols temporarily, which are substituted back at the end. For practical use generating a complete code, `#Optimize` and `#Write` preprocessor instructions are provided in order to give more control to users, which we will not discuss here. The declaration `ExtraSymbols vector,v` specifies the name of the array that will be introduced for storing intermediate results during the evaluation.

The output of the above example is as follows:

```
      v(1)=y*z
      v(2)= - z + 2*y
      v(2)=x*v(2)
      v(3)=z**2
      v(1)=v(2) - v(3) + 2*v(1)
      v(1)=x*v(1)
      v(2)=y**2
      v(2)=2*v(3) + v(2)
      v(2)=y*v(2)
      v(1)=v(2) + v(1)
      F=3*v(1)
*** STATS: original  1P 16M 5A : 23
*** STATS: optimized 0P 10M 5A : 15
```

The numerical evaluation of F needs 18 multiplications and 5 additions, i.e., 23 operations in total, if it is done in the original expanded form. FORM tries to reduce the number of total operations assuming that the cost of addition and multiplications are equal. In the optimized form, the number of operations reduces to 15 operations in total, using the array `v` for temporary variables. By changing the optimization level to `O2`, one will obtain an output with 14 operations. `O3` gives an output with 12 operations.

3. Algorithms

The predefined optimization levels in FORM are summarized as follows:

- `O0`: No optimization.
- `O1`: The occurrence and reverse occurrence orders followed by common subexpression elimination are tried for a Honer scheme.
- `O2`: Besides `O1`, an extra "greedy" optimization is performed to find more common subexpressions at the end.
- `O3`: Monte Carlo tree search is used for finding a good Horner scheme with common subexpressions. At the end the greedy optimization is performed.

Here we will explain each algorithm.

3.1. Multivariate Horner schemes

For optimizing evaluations of univariate polynomials, it is well-known that a *Horner scheme* gives an efficient form for numerical computations:

$$a(x) = \sum_{i=0}^{n} a_i x^i = a_0 + x\Big(a_1 + x\big(a_2 + x(\cdots + x \cdot a_n)\big)\Big). \tag{1}$$

This form takes n multiplications and n additions to calculate a value for a dense polynomial of degree n.

Horner schemes could be extended for multivariate polynomials rather straightforwardly. One can choose one of variables of a polynomial and apply a Horner scheme to it, regarding other variables as constants. Then one can choose another variable from the remaining variables for the next Horner scheme. This is repeated until a Horner scheme is applied to all the variables. However, the number of operations that one can obtain after applying a multivariate Horner scheme depends on in which order the variables are processed.

3.1.1. Occurrence order Traditionally, the *occurrence order* has been often used for multivariate Horner schemes. All variables are ordered in how many times they appear in a polynomial. Applying a Horner scheme to the most-occurring variable gives the largest decrease of the number of operations at each step, because it factors out the most-occurring variable from the polynomial.

On the other hand, one may consider the *reverse occurrence order*. Applying a Horner scheme to the least-occurring variable at each step, the result may contain more common subexpressions within the inner parentheses, which can be detected by *common subexpression elimination* (CSE). Even though this order usually results more operations as a Horner scheme, this order could give fewer operations at the end if it is used in a combination with CSE.

3.1.2. Monte Carlo tree search Both occurrence order and reverse occurrence order may not give the optimal result. For a multivariate polynomial with N variables, the number of possible orders is $N!$. One could try all the orders to find the optimal one when N is small, but obviously it is not practical for large N. Because the orders of the variables can be represented by a *search tree*, one may expect a sophisticated tree search algorithm in computer science could be applied to this problem.

Recently, a new tree search technique, Monte Carlo tree search (MCTS) [7], which was developed in artificial intelligence and game theory, was applied to this problem and it turned out that MCTS actually can give a good order within a reasonable amount of time [5]. Roughly speaking, MCTS picks up some samples from all the orders, at first randomly, but it stochastically gives priority to the neighborhood of good solutions as more samples are collected. It works efficiently if one can expect that good solutions are clustered in branches of the search tree. Note that MCTS is a Monte Carlo algorithm, therefore it may give a different result at each run.

Table 1. FORM run time, gcc compilation time (compile) and the time to evaluate the compiled formula 10^5 times (run) for the resultant with $m = 7$ and $n = 6$. All times are in seconds on an Opteron 2.6GHz processor. The gcc version is 4.6.2.

	Format O0	Format O1	Format O2	Format O3
Operations	587880	71262	55685	36146
FORM time	0.12	1.66	65.43	2398
gcc -O0 compile	29.02	6.33	5.64	3.36
run	119.66	13.61	12.24	7.52
gcc -O1 compile	3018.6	295.96	199.47	80.82
run	24.30	6.88	6.12	3.58
gcc -O2 compile	3104.4	247.60	163.79	65.21
run	21.09	7.00	6.22	3.93
gcc -O3 compile	3125.4	276.77	179.24	71.02
run	21.02	6.95	6.19	3.93

3.2. Common subexpression elimination

In large expressions, the same subexpressions may appear many times. Most of them can be detected and eliminated by a method of *common subexpression elimination* (CSE), which takes time proportional to the input size [8]. Furthermore, for common subexpressions that cannot be detected by CSE, a "greedy" optimization is implemented. This is more or less similar to the algorithm in [9, 10] and much slower than CSE.

4. Benchmark

We performed two kind of benchmark tests of the implemented optimization. One is optimization of the resultant of two polynomials, coming from an example in [11]:

$$\sum_{i=0}^{n} a_i x^i, \qquad \sum_{j=0}^{m} b_j x^j, \tag{2}$$

which is the determinant of a $(m+n) \times (m+n)$ matrix. For this case, we measured the time used in FORM for the optimization, the time used in the compilation and the time to evaluate the compiled formula 10^5 times as well as the number of operations in the optimized code.

Table 1 shows a result for $m = 7$ and $n = 6$, which gives a 13×13 determinant with 43166 terms. One can see that `O2` gives a better optimization than `O1` with respect to the number of operations, and `O3` gives an even better optimization. On the other hand, increasing the optimization level takes more time for the optimization. One can also see that the compilation time as well as the time to evaluate the compiled formula decreases when increasing the FORM optimization level, in all the compiler optimization levels. This result suggests that the best choice of the FORM and compiler optimizations depends on the number of function evaluations needed in the specific problem, but that FORM is more efficient at it than the compiler.

The other benchmark comes from the GRACE system [12, 13], which can generate matrix elements containing one loop diagrams. Each formula corresponds to each diagram and is expressed in terms of Feynman parameters introduced for the loop momentum integration. Within the tensor reduction method used in GRACE, the coefficient of each combination of Feynman parameters must be evaluated separately. In this case the best way is to optimize these coefficients simultaneously, because they may share common subexpressions.

Table 2. Results for three formulas from the GRACE system. For the cases that the number of variables is presented in the form of a sum, the first number is the number of Feynman parameters and the coefficients of combinations of the Feynman parameters were simultaneously optimized. For the left column of HEP(σ), `Bracket` statement was not used. Therefore 35 coefficients with 11 parameters were merged into an expression with 15 parameters. This was done just for comparison, even though it does not make a sense for the GRACE system.

	HEP(σ)	HEP(σ)	F_{13}	F_{24}
Variables	15	4+11	5+24	5+31
Expressions	1	35	56	56
Terms	5717	5717	105114	836010
Format O0	47424	33798	812645	5753030
Format O1	6099	5615	71989	391663
Format O2	4979	4599	46483	233445
Format O3	3423	3380	41666	195691

For this purpose, one can combine two or more expressions into one expression and use the `Bracket` statement just before the optimization:

```
Symbol u;
Local F, G;
Local H = u * F + u^2 * G;
Bracket u;
```

where symbol `u` is used for a label to distinguish the original expressions in `F` and `G`. If `Bracket u` is used, FORM does not perform code optimization with respect to `u`. Therefore the optimized code for `H` still contains `u`. One can split `H` into `F` and `G` by using the exponent of `u`.

For the benchmark test, we picked up three formulas of different sizes. We will call them as HEP(σ), F_{13} and F_{24}. Table 2 shows the result of simultaneous optimizations for the coefficients of combinations of Feynman parameters in these formulas.

In some cases, like expressions appearing in physics, the user can do some work before using the optimization in FORM. One may have knowledge about the problem that can be used for simplification. In the above GRACE example, a set of shifts of variables such as

$$x_i \to x_i + a x_j, \qquad x_i \to x_i + a, \tag{3}$$

can give a significant improvement for the result.

5. Comparison with other programs

We have compared the code optimization implemented in FORM with those of other programs we had access to, with respect to the number of operations in the output. It is summarized in Table 3, which shows that FORM `O3` can give a better result than any other programs.

6. Conclusions

FORM is continuously being developed for bug fixes and adding more features. Recently code optimization was implemented as a new feature. It uses various simplification techniques such as a multivariate Horner scheme, CSE and so on, and is useful when one needs to numerically evaluate lengthy formulas repeatedly or one wants to shrink the sizes of executables. MCTS used in `O3` option gives a better optimization than anything we could find in the literature at some cost of time. This feature will become available as a part of the next version FORM 4.1.

Table 3. The numbers of operations after optimization by various programs. For m-n resultants and a formula appearing in high energy physics (the same one as the left column of HEP(σ) in Table 2). The number for the 7-5 resultant with 'Maple tryhard' is taken from [11]. For the 7-4 resultant they obtain 6707 operations, which must be due to a different way of counting. The same holds for the 7-6 resultant as [11] (HG + cse) starts with 601633 operations. For FORM O3, the parameter C_p was set as $C_p = 0.07$ instead of its default value, and 10×400 tree expansions were used.

	7-4 resultant	7-5 resultant	7-6 resultant	HEP(σ)
Original	29163	142711	587880	47424
FORM O1	4968	20210	71262	6099
FORM O2	3969	16398	55685	4979
FORM O3	3015	11171	36146	3524
Maple	8607	36464	-	17889
Maple tryhard	6451	$\mathcal{O}(27000)$	-	5836
Mathematica	19093	94287	-	38102
HG + cse	4905	19148	65770	-
Haggies [14]	7540	29125	-	13214

References

[1] Kuipers J, Ueda T, Vermaseren J A M and Vollinga J 2013 FORM version 4.0 *Comput. Phys. Commun.* **184** 1453 (*Preprint* 1203.6543 [cs.SC])
[2] Tentyukov M and Vermaseren J A M 2010 The Multithreaded version of FORM *Comput. Phys. Commun.* **181** 1419 (*Preprint* hep-ph/0702279)
[3] Tentyukov M, Fliegner D, Frank M, Onischenko A, Retey A, Staudenmaier H M and Vermaseren J A M ParFORM: Parallel Version of the Symbolic Manipulation Program FORM (*Preprint* cs/0407066)
[4] Tentyukov M, Staudenmaier H M and Vermaseren J A M, 2006 ParFORM: Recent development *Nucl. Instrum. Meth.* A **559** 224
[5] Kuipers J, Vermaseren J A M, Plaat A and van den Herik H J 2012 Improving multivariate Horner schemes with Monte Carlo tree search (*Preprint* 1207.7079 [cs.SC])
[6] Kuipers J and Vermaseren J A M (*in preparation*)
[7] Kocsis L and Szepesvári C 2006 Bandit based Monte-Carlo Planning *LNCS* **4212** 282
[8] Aho A V, Sethi R and Ullman J D 1986 *Compilers: Principles, Techniques, and Tools* (Addison-Wesley)
[9] Breuer M A 1969 Generation of optimal code for expressions via factorization *ACM Commum.* **12** 333
[10] van Hulzen J A 1983 Code optimization of multivariate polynomial schemes: A pragmatic approach *LNCS* **162** 286
[11] Leiserson C E, Li L, Maza M M and Xie Y 2010 Efficient Evaluation of Large Polynomials *LNCS* **6327** 342
[12] Yuasa F *et al* 2000 Automatic computation of cross-sections in HEP: Status of GRACE system *Prog. Theor. Phys. Suppl.* **138** 18 (*Preprint* hep-ph/0007053)
[13] Fujimoto J *et al* 2006 GRACE with FORM *Nucl. Phys. Proc. Suppl.* **160** 150
[14] Reiter T 2010 Optimising Code Generation with haggies *Comput. Phys. Commun.* **181** 1301 (*Preprint* 0907.3714 [hep-ph])