Xtreme manipulations

J.A.M. Vermaseren[*]
Nikhef, Amsterdam
E-mail: t68@nikhef.nl

I discuss a number of extreme situations that the use of FORM has led to. These involve expression sizes, execution times and sizes of generated executable programs. Also the benefits of big computers are discussed. The whole is concluded with speculations about the future.

Loops and Legs in Quantum Field Theory - LL 2014,
27 April - 2 May 2014
Weimar, Germany

[*]Speaker.

1. Introduction

With the running of the LHC and the absence of obvious new physics the need for precision calculations has become only bigger. Also preparations for a new linear collider will need high precision calculations. The purpose of such an ILC will be a look ahead through precision measurements.

It is a well known property of symbolic calculations that intermediate expressions may be many times the size of either inputs or outputs. This is called intermediate expression swell. The capability of a symbolic system to deal will this determines how suitable it is for very large calculations. Of course one may also consider hybrid methods in which only the stages of a calculation in which the intermediate expression swell becomes relevant are dealt with by a system that can handle it, while the other stages are processed by other systems.

Of course, when the expressions become big, the total amount of CPU power will also increase, although the exact relation depends on the nature of the problem and the algorithms used for its solution.

Here we will look at various forms of extreme behaviour.

Disclaimer: I am not claiming that I am the first to run into these problems. It is just that I could only study them by the time I ran into them myself, and then use the experience to improve FORM [1]. One should realize that trying such things out may require enormous resources. Some users have more than I do.

2. Extreme Sizes

Here I am going to look at the tweaking of the Mincer program [2, 3, 4] to allow for the running of rather high Mellin moments (up to $N = 29$). Mincer is a program for evaluating massless three loop propagator diagrams. We use it here for computing Mellin moments of forward scattering in deep inelastic scattering [5, 6]. The state of Mincer a bit over a year ago allowed it to run some of the more difficult diagrams up to Mellin moments N=15, and there it got into trouble. We were running processes in which the external current is taken to be a graviton, as explained in the talk of Andreas Vogt [7]. This means that there are more diagrams, there are many more terms in the projection operators and there are more powers of the momentum P in the projection operators.

Below is one of the projection operators in which terms that should be zero because of gauge invariance have been dropped:

```
Multiply (
    -d_(K1,be1)*d_(K2,be2)*d_(al1,al2)*P.Q^4*Q.Q^-4
    +d_(K1,be2)*d_(K2,be1)*d_(al1,al2)*P.Q^4*Q.Q^-4
    -d_(K1,be1)*d_(K2,be2)*P(al1)*P(al2)*P.Q^2*Q.Q^-3
    +d_(K1,be2)*d_(K2,be1)*P(al1)*P(al2)*P.Q^2*Q.Q^-3
    -d_(K1,be1)*d_(al1,al2)*(P(K2)-Q(K2)*P.Q/Q.Q)*P(be2)*P.Q^2*Q.Q^-3
    +d_(K1,be2)*d_(al1,al2)*(P(K2)-Q(K2)*P.Q/Q.Q)*P(be1)*P.Q^2*Q.Q^-3
    +d_(K2,be1)*d_(al1,al2)*(P(K1)-Q(K1)*P.Q/Q.Q)*P(be2)*P.Q^2*Q.Q^-3
```

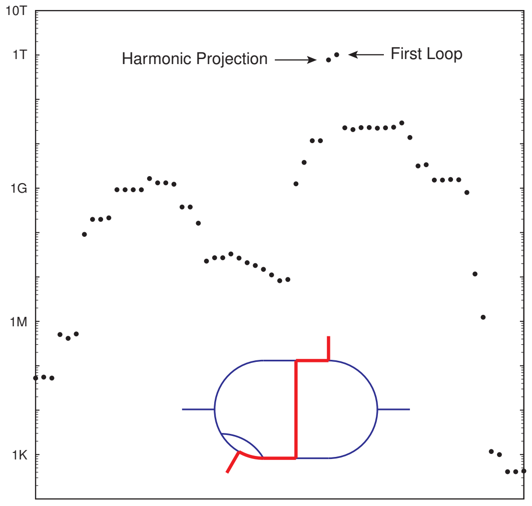

Figure 1: Running profile of an $O2_{26}$ subtopology at N=27.

```
        -d_(K2,be2)*d_(al1,al2)*(P(K1)-Q(K1)*P.Q/Q.Q)*P(be1)*P.Q^2*Q.Q^-3
        -d_(K1,be1)*(P(K2)-Q(K2)*P.Q/Q.Q)*P(al1)*P(al2)*P(be2)*Q.Q^-2
        +d_(K1,be2)*(P(K2)-Q(K2)*P.Q/Q.Q)*P(al1)*P(be1)*P(al2)*Q.Q^-2
        +d_(K2,be1)*(P(K1)-Q(K1)*P.Q/Q.Q)*P(al1)*P(al2)*P(be2)*Q.Q^-2
        -d_(K2,be2)*(P(K1)-Q(K1)*P.Q/Q.Q)*P(al1)*P(be1)*P(al2)*Q.Q^-2
)*4/Q.Q;
  #define PROJ "4"
```

The problem was that Andreas needed a few more moments to produce nice physics. Hence the Mincer code needed to be improved, because at the time it was optimized for moments in the range of 10 to 12 with simpler projection operators. Examples of running profiles (a histogram of the expression size at the end of each module) for various (sub)topologies are shown in Figs 1, 2, 3:

These examples show that each topology and each subtopology has to be optimized individually. This is quite some work.

Some of the generic improvements that were made to get this far were

A Put diagrams with the same subtopology, the same color factor and the same flavor structure together as if they are a single diagram. It turns out that this combined set usually takes not much more time than the most expensive diagram in the set. The gain in time is a factor in the range of 3 and 5.

B Spend some effort in rerouting the P flow through the diagram. It is very important that the number of propagators to be expanded is minimal.

C Group the recursions in such a way that there are neither too few nor too many of them in one module. This requires experimentation.

D Make sure that momentum substitutions are economical.

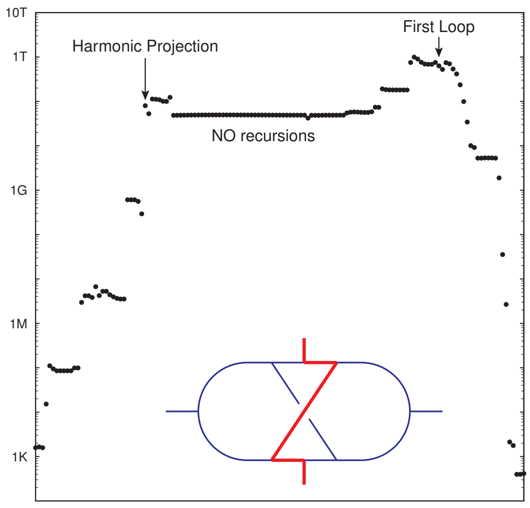

Figure 2: Running profile of a NO_{25} subtopology at N=27.

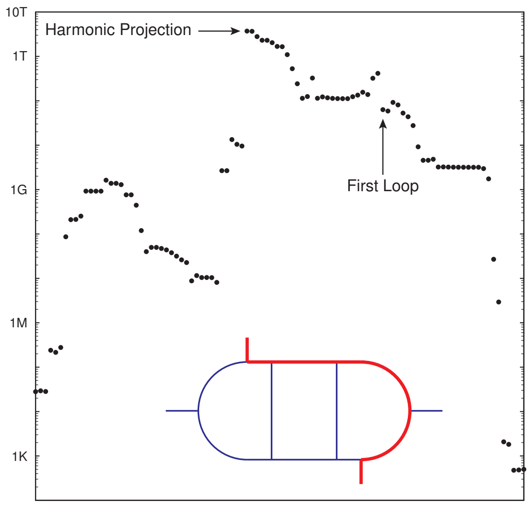

Figure 3: Running profile of a LA_{14} subtopology at N=29.

Ad A: Kostia Chetyrkin and collaborators have been doing this for years with even the color factors worked out for SU(3) [8] so that they become numbers and even more diagrams can be grouped together. In the first Mincer calculations this was not done, because the concept of subtopologies was not well developed and mixing different subtopologies gave severe space problems. Also Form was less flexible in those days. Hence diagrams were treated one by one with the spinoff that color factors were in terms of group invariants.

Ad B: The selection of which propagators are the ones that need to be expanded is critical. Often rewriting of the momenta can help very much as illustrated in Fig 4.

This is particularly important for the diagrams that peak during the harmonic projection or shortly after, but the non planar recursions are also very sensitive to this.

Ad C: If the full reduction takes for instance 30 steps and we do them all in a single

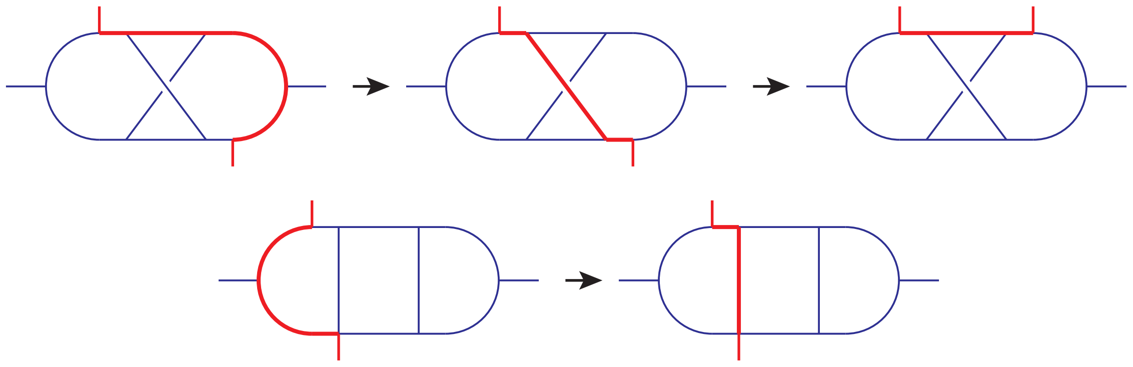
Figure 4: Transformation rules for the P-momentum flow (red line).

module, we may generate an astronomical number of terms. The sorting will take care of this, but the step may take way too much time. If in each module we do a single step, there will be too much sorting of terms that were not affected. The optimum is to to do a number of recursion steps in each module. The precise number needs to be obtained by experimentation.

Ad D: The vector substitution

```
id p1 = p3-p5;
```

can cause problems, because powers of dotproducts will be written out without the use of binomial coefficients. Hence `p1.p2^20` would generate 2^{20} terms. Writing

```
id p1.p1 = p3.p3-2*p3.p5+p5.p5;
id p1.p? = p3.p-p5.p;
```

is far more economical, because now FORM can use binomial coefficients.

3. Extreme Times

The following is the final line of a program that proved a critical identity for Multiple Zeta Values. It confirmed a guess about the basis for MZV's of weight 27 and showed the first double pushdown which had been predicted in the paper about the Multiple Zeta Value datamine [9, 10].

```
171258.46 sec + 55845418.93 sec: 56016677.39 sec out of 7345664.84 sec
```

This is a rather impressive time: 85 days (648 days of CPU time). It was the biggest computer I could lay my hands on at the time (2010). The program had to be run like this, because we (Jan Kuipers and I) could not find a way to cut the job in pieces. Actually, it was already the result of cutting a bigger program in 9 pieces. This was the biggest irreducible piece.

We run here into a problem that is all too familiar to some people: if you need to run a job this size and you do not have a big enough computer to run this 'in house', you need to use the equipment of a computer center. Usually they will not allow jobs that take this long. They will definitely not garantee more than a month uptime.

There are several ways to tackle this problem.

1. Have the computer center get a more powerful computer.
2. Get the money to buy an adequate 'in house' computer.
3. Find a way to cut up your job further.

One way to have a more powerful computer is to have more cores. We will come to that later.

Fortunately, in principle, FORM has a solution provided you have enough diskspace. It is called the checkpoint facility in which one can make, at the end of each module, a safety backup of the current state. Only one such backup is made, which means that older ones will be removed automatically. Hence, if your program crashes because the computer was turned off, you can restart it later at the beginning of the module in which it was at the moment of the crash. Of course, it will cost some extra time to write such a backup if your program uses much memory and large files. Now, when the computer center kills your job, you can restart it with hopefully not too big a loss.

Unfortunately I am not aware of people having tried out this checkpoint facility extensively. Hence, my guess is that for now it may still contain errors. When Jens Vollinga created it, it worked on all his examples, but whenever there are new features, the corresponding variables have to be added and it is not clear that this has always been done correctly. It is just that people should be aware of this possibility, and if they try it and there are indeed problems the FORM team will do its best to resolve those as soon as possible.

Of course, if one is already using most of the disks, there may not be enough space for the backup files. In that case one may have to look at the other possibilities mentioned above, with the remark that a bigger disk is often the cheapest and easiest solution.

4. Extreme Executables

Recently I had a masterstudent (Otto Rottier) who's task it was to make some extra software for the GRACE [11] system and then try it all out on some 'simple' reactions like $e^+e^- \to ZZh$ or $e^+e^- \to Zhh$. He was also to provide some manuals in the form of a master thesis [12].

It should be known that systems like the GRACE system are complex and when one tries new things one may still run into cases that have not been considered before, Hence there may be missing or incomplete code.

We ran into a gauge problem in the second reaction. Fortunately this reaction has been computed before. This was however with the old REDUCE [13] version of GRACE [14]. The task became now to compare the two. To make things easier we removed the vertices in which the electrons couple to scalars. This brings the number of loop diagrams down by

Type of Files	Lines	Size
.red	6,040,575	121,719,873
.f .F	39,988,122	2,604,161,410
.o		19,290,341,352
test0		8,???,???,???

Table 1: File sizes with the REDUCE version of GRACE. Fortran compilation was with -O0. The total running time was a bit more than 8 hours. .red are the REDUCE files, .f and .F the Fortran files and .o the object files after compilation. Test0 indicates the size of the executable.

Type of Files	Lines	Size
.frm	1,296,681	27,126,191
*.f *.F	5,624,124	149,450,518
*.o (-O0)		713,971,472
test0 (-O0)		333,242,881
*.o (-O3)		163,567,952
test0 (-O3)		103,879,860

Table 2: File sizes with the FORM version of GRACE. Fortran compilation was with -O0 and -O3. The total running time was a bit more than 2 hours. .frm are the FORM files, .f and .F the Fortran files and .o the object files after compilation. Test0 indicates the size of the executable.

almost a factor 3 and the number of tree graphs by an even bigger factor. We first checked in the FORM version that these diagrams were not the cause of our problems.

All runs were on a computer with 24 cores. The code was compiled in double precision with ifort, the Intel Fortran compiler. The tables 1, 2 show some properties of the resulting code and programs.

What one sees here is:

- The more sophisticated output optimizations of FORM (procedures to compactify the expressions in combination with the built in output optimizations [16]) are almost a requirement.
- By the time we start doing $2 \to 4$ reactions also the FORM executables will exceed the magic 2 Gbytes limit. Programs will need to be split up.
- It seems impossible to compile the REDUCE output with any type of optimization. Already at -O0 some files take more than 2 hours on this fast processor.
- The FORM optimization prepares the output very well for the compiler. The total compilation time was of the order of 5 minutes, even with the -O3 option.

At the moment a graduate student (Ben Ruyl) is working on improving the optimizations even further and make FORM execute them much faster.

The splitting of the programs can be done in several ways. The original approach in the GRACE system is to split the matrix element in pieces, each in a separate program.

Weight	workers	CPU Time at end	Real Time
20	-	41265.26 sec	41300.92 sec
20	0	43219.45 sec + 0.00 sec: 43219.45 sec	43250.09 sec
20	1	11778.75 sec + 36006.46 sec: 47785.21 sec	47026.03 sec
20	2	2156.57 sec + 44627.86 sec: 46784.44 sec	24455.02 sec
20	4	2122.24 sec + 47252.28 sec: 49374.52 sec	14054.96 sec
20	8	2402.49 sec + 50515.10 sec: 52917.60 sec	8951.90 sec
20	12	2496.42 sec + 54214.88 sec: 56711.31 sec	7256.94 sec
20	16	2570.70 sec + 56982.59 sec: 59553.29 sec	6304.62 sec
20	20	2637.25 sec + 62331.76 sec: 64969.01 sec	6035.57 sec
20	24	2628.24 sec + 66821.74 sec: 69449.98 sec	5722.42 sec

Table 3: Real time as a function of the number of workers. A dash indicates sequential FORM. A zero means that in TFORM the master does all the work.

All these programs are then commanded from a single master program that makes each of them evaluate their pieces of the matrix element for identical Monte Carlo points and return their values to the master. The master adds them, evaluates the efficiency of the integration and decides on a new set of Monte Carlo points for all workers. This way the gauge cancellations are not polluted by the Monte Carlo statistics. This method and a new implementation of the software for it is described in more detail in the master thesis of Otto Rottier [12].

The newer approach is to make sets of diagrams into position independent library files that can be dynamically linked (.so files). Although a complete executable with all diagrams in it cannot exceed 2 Gbytes (limitation by omission in the GNU software), this seems to work well even though loading the program can be slow.

Each method has advantages and disadvantages.

5. Extreme Computers

From the previous sections it should be clear that any increase in power of symbolic systems immediately makes more problems solvable. I will concentrate on FORM of course, but it may be noted that recently Maple greatly enhanced its polynomial performance by creating specialized code for dealing with polynomials [15]. The interesting part is that its internal notation has evolved a little bit in the direction of the FORM notation. Because it is completely programmed in C and treats only polynomials, it is rather fast.

One way to improve the performance of FORM is to improve its parallel capabilities. This is not as easy as it seems. Table 3 shows how using more cores can give saturation effects.

This can only be improved up to a point. There will always be an entity that is in charge of the 'whole', but in my viewpoint there are ways to defuse most of the bottlenecks. This will however involve more research in methodology.

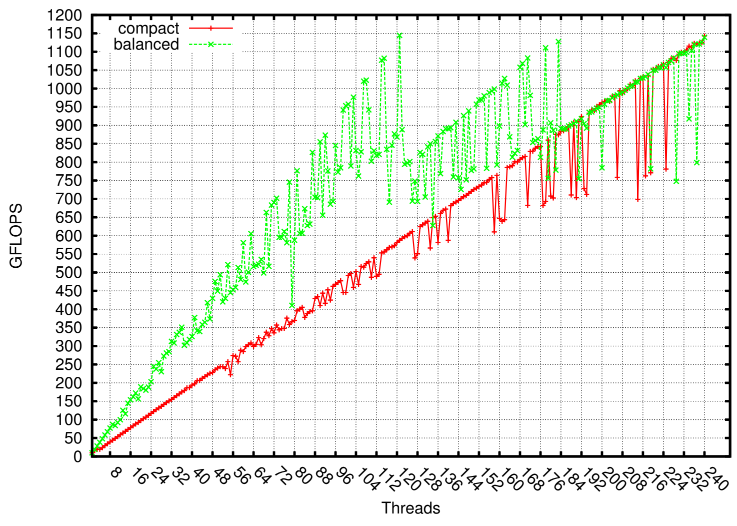

Figure 5: Performance of a Phi accelerator board on a floating point benchmark.

A second step involves massive numbers of cores on coprocessor boards. In this case one should abandon the concept that we should use the available CPU resources completely. If one has one or more boards with thousands of cores, one has basically a near infinite amount of CPU power as compared to the capabilities of getting information in the right place at the right time. The data transfer will become the big bottleneck. If this is all to take place inside a single computer, bus congestion and cache use may become a frequently used words. In that case a good network and many individual computers (each with a number of cores) would become interesting. This is why both the TFORM [17] and the ParFORM [18] concepts are viable, even though at the moment TFORM is easier to use.

At the moment we have started to experiment with the Intel Phi accelerator board. This is the task of a new graduate student Ali Mirsoleimani. Fig. 5 shows one of his benchmarks. Of course we cannot use vector pipes very well in FORM, because about 20% of its C-code involves if-statements, but we should be able to pick up some of this nice speed.

The third step involves the development of new algorithms to do calculations. Algorithms that allow formulas to be processed in a distributed way without much loss in performance. If we go back to the graph of Fig. 1 with the two steps that blow up the formula, one with a collapse of a factor 40 (number of generated terms divided by number of terms in the output) and the second with a collapse of a factor 6, it is possible to split up the calculation, provided one can separate the input expression in such a way that there is not much crosstalk between the pieces. This is not a simple task.

6. Conclusions

Because there is a need for big calculations, it is important to increase capabilities in that direction. These directions include

- Better space control.

- Faster code.
- Better use of multiple cores.
- More compact outputs.
- More sophisticated algebraic capabilities.

Acknowledgements: Some of the programs I have shown here were run on computers at DESY (Zeuthen), Karlsruhe (KIT) and Liverpool. The others were run at Nikhef with computers, some of them paid for by the ERC advanced grant 320651, "HEPGAME". Work in the various examples was done while staying at Nikhef, DESY, KIT, KEK and UAM-Madrid, and in a variety of collaborations.

References

[1] J.A.M. Vermaseren, *New features of FORM*, math-ph/0010025. The program can be obtained from https://github.com/vermaseren/form.

[2] S.G. Gorishnii, S.A. Larin and F.V. Tkachov, INR preprint P-0330 (Moscow, 1984).

[3] S.G. Gorishnii, S.A. Larin, L.R. Surguladze and F.V. Tkachov, *Mincer: Program for Multiloop Calculations in Quantum Field Theory for the Schoonschip System*, Comp. Phys. Comm. 55 (1989) 381.

[4] S.A. Larin, F.V. Tkachov and J.A.M. Vermaseren, *The FORM version of MINCER*, NIKHEF-H-91-18.

[5] S.A. Larin, Paulo Nogueira, T. van Ritbergen and J.A.M. Vermaseren, *The Three loop QCD calculation of the moments of deep inelastic structure functions*, Nucl. Phys. B492 (1997) 338-378, hep-ph/9605317.

[6] J. Blümlein and J.A.M. Vermaseren, *The 16th moment f the non-singlet structure functions $F_(x,Q^2)$ and $F_L(x,Q^2)$ to $\mathcal{O}(\alpha_S^3)$*, Phys. Lett. B606 (2005) 130-138, hep-ph/0411111.

[7] A.Vogt, S. Moch and J.A.M. Vermaseren, *A calculation of the three-loop helicity-dependent splitting functions in QCD*, arXiv:1405.3407.

[8] K.G. Chetyrkin, *Quark mass anomalous dimension to $\mathcal{O}(\alpha_S^4)$*, Phys.Lett. B404 (1997) 161-165, hep-ph/9703278.

[9] J. Blümlein, D.J. Broadhurst and J.A.M. Vermaseren, *The Multiple Zeta Value Datamine*, Comput. Phys. Commun. 181 (2010) 582-625. arXiv:0907.2557(math-ph).

[10] J.Kuipers and J.A.M. Vermaseren, *About a conjectured basis for Multiple Zeta Values*, arXiv:1105.1884(math-ph).

[11] J. Fujimoto et al. *GRACE with FORM*, Nucl. Phys. Proc. Suppl. 160 (2006) 150-154.

[12] Otto Rottier, *One loop corrections at future colliders*, master thesis, in preparation.

[13] A.C. Hearn, *REDUCE 2: A System and Language for Algebraic Manipulation.*, Proc. 2nd Symp. on Symbolic and Algebraic Manipulation, ACM New York (1971), pp. 128-133. See also www.reduce-algebra.com and reduce-algebra.sourceforge.net.

[14] G. Belanger et al. *Full $\mathcal{O}(\alpha)$ electroweak corrections to double Higgs strahlung at the linear collider*, Phys. Lett. B576 (2003) 152-164.

[15] Michael Monahan and Roman Pierce: *POLY: A new polynomial data structure for Maple 17*, ACM Communications in Computer Algebra, Vol 46, No. 4, Issue 182, December 2012.

[16] J. Kuipers, T. Ueda and J.A.M. Vermaseren, *Code Optimization in FORM*, arXiv:13107007(cs.SC).

[17] M. Tentyukov and J.A.M. Vermaseren, *The Multithreaded version of FORM*, Comput. Phys. Commun. 181 (2010) 1419-1427, hep-ph/0608307.

[18] M. Tentyukov et al. *ParFORM: Parallel version of the symbolic manipulation program FORM*, cs/0407066.