What is new in FORM

J.A.M. Vermaseren[a] and M. Tentyukov[b*]

[a]NIKHEF,Kruislaan 409, 1098SJ Amsterdam, The Netherlands

[b]Institut für Theoretische Teilchenphysik, Universität Karlsruhe D-76131 Karlsruhe, Germany

We describe new developments in the symbolic manipulation system FORM. Particular attention is given to the implementation of parallel processing of formulas.

1. Introduction

FORM[1] has usually seen lots of additions when new calculations were undertaken. These additions then became part of the official version either during or shortly after the calculation was finished.

Last year a big calculation[2–4] was finished which involved lots of new facilities in FORM. Many made it already into version 3.1 and were reported on at earlier workshops. But the period after a big calculation is very good for working at FORM and putting in facilities that were overdue. Also the people in Karlsruhe had some needs and M. Tentyukov has been building in features for running FORM together with other programs and having them communicate with each other[5].

One problem is proper testing, and another is to make a proper abstraction of the needs of the calculations. This abstraction means that commands should not be specific for a single problem, but everybody should benefit from it. Because of this a period is needed for experimentation. This means that it will take a while before the new features become available and that the first released versions may contain undesirable properties, because each user applies the facilities in private ways. Hence the importance of interaction with the users cannot be underestimated.

In this talk we describe various facilities for output reduction, the application of the zlib library to compress the intermediate sort files, we mention the new channels for running external commands, after which we switch to the main topic of parallel processing with ParFORM and TFORM.

2. Various commands for output reduction

Sometimes we produce lengthy outputs for further numerical evaluation in a FORTRAN program. When these outputs become more than a few thousand lines the compiler may actually start giving problems and the question arises whether we can simplify these lengthy FORTRAN outputs. This is a science by itself. There are various approaches:

- Find another program that can do the job and feed it the expression. This may not be easy due to the size of the expression.
- Split the expression in pieces and try again. This is not optimal, but it may improve things very much already.
- Try to do this within FORM by looking for common subexpressions. This is also far from ideal, but at least it is not limited by the size of the expressions.

The first two methods can use the new external channels (see later). There are various new functions that can help in the third method. They have been developed for the GRACE effort[6]. Lack of space prevents us from showing explicit examples here. We will just mention the new command 'MakeInteger' and the functions 'nterms_' and 'termsinbracket_'.

*On leave from the Joint Institute for Nuclear Research, 141980 Dubna, Moscow Region, Russian Federation.

The MakeInteger command normalizes the argument(s) of the specified function(s) in such a way that all terms in the argument have an integer coefficient and their GCD is one. A potential normalization factor is multiplied into the coefficient of the term with a given power.

The function 'nterms_' is replaced by a number which indicates the number of terms that the function had for its argument.

The function 'termsinbracket_' is to be used together with brackets and the 'Keep brackets' statement. It is replaced by the number of terms that are in the bracket that is currently treated.

Together the above features allow already the construction of programs that managed to reduce the FORTRAN outputs of the GRACE/1-loop program by a factor four in the test programs[6].

3. GZIP

The maximum size of expressions that can be handled by FORM is determined by the available file space. When there is only a single expression the total file space needed during the calculation is one scratch file for input or output and the sort file. The size of the sort file will always be at least the size of the output scratch file, but it can be considerably larger.

The exists already some form of data compression in the sort and scratch files, but it uses a rather simple method and it is not nearly as efficient as the well-known compression utility GZIP. It has the advantage though that it allows FORM to select pieces of the expression like specific brackets, whereas GZIP doesn't allow the extraction of pieces from the middle of the file.

When GZIP is applied to these files one can obtain compression ratios that are typically a factor 4, although this will depend very much on the problem and the hardware. If there are mainly very lengthy coefficients, the ratio will be less. If one works on a 64-bits computer and the coefficients are short numbers, the ratio will be larger.

Currently the sort file can be treated with the gzip library. Each sorted 'patch' is a gzipped stream and when the file-to-file sort takes place they are simultaneously gunzipped slowly to feed the merging routine.

The drawback is that using GZIP costs CPU time. Unfortunately experimentation shows that this isn't gained back by the fact that now less writing to files and reading from files is needed. Hence one should use this facility mainly when one is really short of disk space.

Syntax:

```
On compress GZIP 6;
```

This selects compression level 6 which seems a nice compromise between speed and compression. The highest level is 9, but this is very slow and doesn't compress much better than level 6. Level 0 means no compression.

It would have been possible to use bzip, which compresses better. It is however extremely slow. Maybe in the future a better algorithm will be developed that takes more the structure of a FORM binary file into account. The future may also see compression of the scratch files, but that will lead to a reduction of the possible commands that can be used in the next module as 'keep brackets' and the obtaining of bracket contents will not be possible.

4. External channels

Communication with external programs existed in FORM already in a relatively simple way. The implementation of the Laporta algorithm[7] at the university of Karlsruhe[8] showed the need for a more sophisticated method of communication. During evaluation many subexpressions are obtained that are sums of rational polynomials in d, the dimension of space-time. These expressions need to be simplified by giving the terms a common denominator and then dividing out the GCD. For this they use the program FERMAT[9].

The resulting implementation of the external channels is described in a very recent paper[5]. Here we show just an example that sends the expression withGCD to FERMAT and picks up the answer. We do this 1000 times just to show the speed.

```
Off statistics;
Format 250;
#define cmd "./fermat"
#define init "&d\n0\n&(M=\' \')
```

```
        \n&(t=0)\n&U\n&(J=d)\n"
Symbol d;
#external 'cmd'
#toexternal "'init'"
#do i = 1,19
  #fromexternal "tmp"
#enddo
Local withGCD =(2*d^4+3*d^3-22*d^2
         -13*d+30)/(d^3-11*d+10);
.sort
#do i=1,1000
  #toexternal "%E\n",withGCD
  #fromexternal "tmp"
  Local noGCD =
  #fromexternal
           ;
  Print;
  .sort
  Drop noGCD;
  .sort
#enddo
.end
```

The tail of the output file is:

```
   noGCD =
      3 + 2*d;

     .end
real  0m0.799s
user  0m0.160s
sys   0m0.010s
```

It should be clear that there is much room for creativity here. Sometimes most of the work is in building the filters/gateways that translate the notations of the various programs or manage information of programs in such a way that the others can work with it.

5. ParFORM

ParFORM is a project at the university of Karlsruhe, funded by the DFG. People who have worked at it in the past are H.M. Staudenmaier, A. Retey, D. Fliegner, A. Onyshenko, M. Frank. Currently M. Tentyukov and M. Steinhauser are in charge of it. The project has led to a number of publications[10–12]. As seen in the above section the dexterity of M. Tentyukov with the FORM sources also gives us new features. ParFORM is a version of FORM that uses several processors or computers, connected by a network, in a way that makes it look to the user as a single much faster computer.

ParFORM keeps getting improved. Currently its main machines are SMP SGI computers at Karlsruhe with eight and 32 Itanium II processors at 1300 MHz. Several physical results have been obtained in the recent years[13] dealing mainly with four-loop propagator-type diagrams. Calculations performed on a group of eight processors are up to six times faster than those on a single processor, while the use of 16 processors leads to a speedup factor of ten. Beyond 16 processors the speedup curve becomes flatter, however, for some applications also this region is of interest in order to manage the evaluation of very complicated expressions.

Recently, ParFORM has been successfully installed on dual Opteron clusters both interconnected via Gigabit Ethernet and InfiniBand. A reasonable speedup behaviour of 4-5 has been observed for 8 processors.

ParFORM also runs on several other computers. The problem with the ports is that there is no rigorous standard for the MPI libraries. This means that each port is much work. They are made on demand and availability of time and resources. As ParFORM exists basically already for a number of years, we will not go into its specifics in the context of this talk.

6. TFORM

This is a rather new project. The aim is to make a multi-threaded version of FORM that can run on computers with shared memory. In that case the data is divided in shared global data and thread specific local data. This way much less data has to be transfered. On the other hand: the routines have to be 'thread-safe'. Currently this is where the major source of errors is.

The program is currently being debugged and tuned. Already some improvements have been made and MINCER is already running with it. The main machine for experimentation is a quad opteron machine with 4 opteron 850 processors at 2.4 GHz and 16 GBytes of memory.

The running of TFORM is rather simple:

```
tform -w4 calcdia
```

would run the program calcdia.frm with 4 worker threads and one master thread. The command:

```
tform calcdia
```

would just run everything inside a single master thread. No workers will even be started. This is on average slightly slower (2-3%) than the sequential version.

The treatment of threads follows the Pthread (POSIX threads) standards which are widely available, definitely in UNIX systems. Hence there are very few problems with ports.

There are some features that allow the user to tune the system a bit. Terms are send to the workers in groups or 'buckets'. The optimal size of the buckets depends on the problem.

Sometimes all nasty terms can be in one bucket making one worker doing all the work. The master and the other workers have to wait for this worker to finish. There is a loadbalancing system in which the master can steal the unprocessed contents of the bucket of a thread that is still busy at the end of the module. When it isn't needed it can be switched off. A second stage load balancing is being planned. This would be for when there is a single bad term. In that case a worker can give branches of the expansion tree to other workers.

There will be a separate paper about version 3.2 of FORM that will explain more of the technical details. Here we will show some results. The computers that are being used are

- alfonso: a Dual Pentium IV at 1.7 GHz.
- norma: a Quad opteron system with 4 Opteron 850 processors at 2.4 GHz.
- The use of hyperthreading processors gives only a relatively small increase of the work that can be performed, which is mostly offset by the extra overhead. Here TFORM would only be useful in crowding out other programs. We will not consider them here.

The first example program we consider is from a suite of examples that was constructed several years ago on the suggestion of D.Fliegner. They concern the construction of so-called chromatic polynomials on a lattice. The whole project from the viewpoint of FORM is to show how one can successively improve the efficiency of a program. The whole paper and the programs can be found in the FORM distribution site. The crucial part of the program is

```
#do i = 0,'N'^'D'-1
  Multiply F'i';
  repeat id  d(?a,k?,?b)*d(?c
       ,k?,?d) = d(?a,?c,k,?b,?d);
  Symmetrize d;
  repeat id  d(?a,k?,k?,?b) =
                      d(?a,k,?b);
  #do d = 1,'D'
    id  d(k'i',?a,k1?kk0[x],
           ?b,k2?kk'd'[x],?c) = 0;
  #enddo
  id,ifmatch->1,d(k'i',k?) = 1;
  id,ifmatch->1,d(k'i',?b) = d(?b);
    Multiply acc([q-1]+1);
  Label 1;
  id  d = 1;
  .sort:'i';
#enddo
```

The complete program can be obtained from the FORM distribution site once version 3.2 of FORM will be available.

We run the program for a 13x13 lattice. The improvement factor is the improvement in the real time that it takes to run the program. When we mention 0 workers we refer to the sequential version of FORM. For one worker we refer to TFORM in which the master runs the whole program. This shows part of the overhead. There is more overhead when the workers are actually used, because now signals have to be sent and information has to be copied to and from the workers. The results can be seen in the tables 1 and 2. We notice that running more than two workers on a two processor system (table 1) gives a slight penalty and makes the improvement go down a bit. But it is very well possible to run with more threads than there are processors.

The next example concerns some diagrams that are run with MINCER[14,15]. We selected 3 diagrams: d1c, d10c and d11c from a dataset for

Workers	CPU-time	real time	CPU-master	improvement
0	1871.57	1921.14		
1	1908.60	1944.34		
2	2021.47	1089.56	159.51	1.7632
3	2041.79	1115.65	192.73	1.7220
4	2057.47	1141.83	216.61	1.6825

Table 1: Program p15 running on a Dual P4(1700).

Workers	CPU-time	real time	CPU-master	improvement
0	710.15	710.69		
2	738.12	386.17	50.17	1.8403
4	788.88	244.00	70.40	2.9127

Table 2: Program p15 running on a Quad Opteron 850 (2.4 GHz).

calculating nonsinglet contributions to photon-quark scattering. These are all non-planar diagrams. The first is moderately simple and we can calculate the N=16 moment. The second diagram is moderately difficult and the third diagram is the most difficult of the set. Of the last two diagrams we calculate the N=10 moment. The results can be seen in the tables 3 and 4.

In big calculations the major weakness is at the end of the module when all workers have to collect the contents of their sort file, merge them and send the results to the master thread which merges these streams into a single output file. The resulting traffic jam can bring processing almost to a halt. Sometimes it can make running on 4 processors slower than on a single one. What to do about this is still being investigated as the situation of the disk on the quad machine was far from ideal. A better disk has improved things considerably.

There remains the issue that is there is much writing activity, this will slow things down. The example used was the p15 program with a 15x15 lattice. In the program about 250 GBytes of files had to be written when 4 workers were used. In the sequential version this is less as the buffers can be bigger. Hence 'only' 139 GBytes have to be written. The result is that, even with 4 threads and much optimization concerning synchronizing the file system, TFORM is only a little bit faster (see table 5): It should be clear that considerations of buffer sizes are important when selecting how to run big programs.

Workers	CPU	real	GBytes
0	11614	14210	139
4	13974	10535	250

Table 5: Program p15(15x15) running on a Quad Opteron 850 (2.4 GHz).

7. Conclusions

The new developments in FORM will open new ways of doing symbolic calculations. Unfortunately the debugging has still not been completed. As soon as the number of problems gets so low that we don't see any, we will make this version available. Most likely programs of other people will run into problems. Please report those in as concise a way as possible in such a form that we have a chance to reproduce them. Debugging multi-threaded programs is rather complicated and hence one needs as much information as possible.

REFERENCES

1. J. A. M. Vermaseren, *New features of FORM*, math-ph/0010025
2. S. Moch, J.A.M. Vermaseren and A. Vogt, Nucl. Phys. B688 (2004) 101, hep-ph/0403192
3. A. Vogt, S. Moch and J.A.M. Vermaseren, Nucl. Phys. B691 (2004) 129, hep-ph/0404111

Diagram	Workers	CPU-time	real time	CPU-master	improvement
d1c	0	2675.80	2676.18	-	-
d1c	1	2729.62	2729.93	-	
d1c	2	2804.31	1433.28	45.53	1.8672
d1c	3	2810.25	1438.95	66.43	1.8598
d10c	0	2649.63	2651.87	-	-
d10c	2	2796.84	1440.82	77.47	1.8405
d11c	0	10344.22	10400.52	-	-
d11c	2	10870.10	5693.18	333.51	1.8268

Table 3: MINCER diagrams running on a Dual P4(1700).

Diagram	Workers	CPU-time	real time	CPU-master	improvement
d1c	0	754.05	755.73		
d1c	1	807.78	808.88		0.9343
d1c	2	827.36	424.80	15.67	1.7790
d1c	3	833.22	291.09	21.67	2.5961
d1c	4	835.19	228.79	26.72	3.3031
d10c	0	851.12	852.09		
d10c	2	928.02	471.77	30.21	1.8062
d10c	4	945.15	262.03	38.29	3.2519
d11c	0	3124.07	3125.05		
d11c	2	3408.98	1720.59	107.18	1.8163
d11c	4	3513.83	984.07	136.36	3.1756

Table 4: MINCER diagrams running on a Quad Opteron 850 (2.4 GHz).

4. J.A.M. Vermaseren, A. Vogt, and S. Moch, hep-ph/0504242 (Nucl. Phys. B, in press)
5. M.Tentyukov and J.A.M. Vermaseren, cs-sc/0604052
6. J. Fujimoto at al., these proceedings.
7. S. Laporta, Int.J.Mod.Phys.**A15**(2000)5087; hep-ph/0102033
8. K. G. Chetyrkin *et al*, Eur.Phys.J. **C40**(2005) 361-366; K.G. Chetyrkin, J.H. Kühn and C. Sturm, hep-ph/0512060; K.G. Chetyrkin *et al*, hep-ph/0601165.
9. http://www.bway.net/˜lewis
10. M. Tentyukov, D. Fliegner, M. Frank, A. Onischenko, A. Retey, H. M. Staudenmaier and J. A. M. Vermaseren, arXiv:cs.sc/0407066.
11. D. Fliegner, A. Retey and J. A. M. Vermaseren, arXiv:hep-ph/0007221.
12. D. Fliegner, A. Retey and J. A. M. Vermaseren, arXiv:hep-ph/9906426.
13. Some recent results are found in:
P. A. Baikov and K. G. Chetyrkin, arXiv:hep-ph/0604194; P. A. Baikov, K. G. Chetyrkin and J. H. Kuhn, Nucl. Phys. Proc. Suppl. **157** (2006) 27 [arXiv:hep-ph/0602126].
14. S. G. Gorishnii, S. A. Larin, L. R. Surguladze and F. V. Tkachov, Comput. Phys. Commun. **55** (1989) 381.
15. S. A. Larin, F. V. Tkachov and J. A. M. Vermaseren, NIKHEF-H-91-18