PROCEEDINGS OF SCIENCE

Parallel versions of the symbolic manipulation system FORM

M. Tentyukov*
Institut für Theoretische Teilchenphysik, Karlsruhe Institute of Technology (KIT), D-76128 Karlsruhe, Germany
E-mail: tentukov@particle.uni-karlsruhe.de

J.A.M. Vermaseren
Nikhef Science Park 105 1098 XG, Amsterdam
E-mail: t68@nikhef.nl

J. Vollinga
Nikhef Science Park 105 1098 XG, Amsterdam
E-mail: jensv@nikhef.nl

The symbolic manipulation program FORM is specialized to handle very large algebraic expressions. Some specific features of its internal structure make FORM very well suited for parallelization.

We have now two parallel versions of FORM, one is based on POSIX threads and is optimal for modern multicore computers while another one uses MPI and can be used to parallelize FORM on clusters and Massive Parallel Processing systems. Most existing FORM programs will be able to take advantage of the parallel execution without the need for modifications.

13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research
February 22-27, 2010
Jaipur, India

*Speaker.

1. Introduction

The symbolic manipulation system FORM [1] which is available already more than 20 years, is specialized to handle very large algebraic expressions of billions of terms in an efficient and reliable way. It is widely used, in particular in the framework of perturbative Quantum Field Theory, where sometimes hundreds of thousands of Feynman diagrams have to be computed; most of the spectacular calculations of refs [2, 3] would hardly have been possible with other available systems. However, the abilities of FORM are also quite useful in other fields of science where the manipulation of huge expressions is necessary.

Parallelization is one of the most efficient ways to increase performance. Some internal specifics [4] make FORM very well suitable for parallelization so the idea to parallelize FORM is quite natural.

2. General concepts and models in use

The general concept of FORM parallelization is as follows [4, 5, 6]: upon the startup, the program launches a *master* and several *workers*. FORM treats each expression individually, which allows the master to split incoming expressions into independent chunks. Each chunk is processed by workers in parallel, and then the master collects the results.

At present, we have two different models [5, 6]: in `ParFORM` [4] the master and workers are independent processes communicating via MPI[1] and in `TFORM` [6] master and workers are separate threads[2] of a multithreaded process.

Both models require almost no special efforts for parallel programming, all FORM programs may be executed in parallel without any changings. The user may give FORM some hints of how to parallelize some things better; these hints are simply ignored by the sequential version of FORM.

Since TFORM uses common address space, it is runnable only on SMP computers. On the other hand, sometimes it permits more efficient parallelization, and it does not depend on MPI which make it much easier for deployment. ParFORM can be used not only on SMP computers but also in clusters and Massive Parallel Processors (MPP).

3. Performance

Both ParFORM and TFORM demonstrate approximately the same speedup [5, 6]. Here we discuss TFORM running the Multiple Zeta Value program [7] on the computer “qftquad5” at DESY. The computer has 96 GB of main memory and 8 independent CPU cores; the effective number of CPU cores is 16 due to hyperthreading. The results are given in Fig. 1.

For reference, the run with FORM (the sequential version) took 57078 sec.

We see three regions: first, the speedup is almost linear up to 8 workers; second, the speedup is also almost linear in the range of 8-16 workers but with much less slope, and after 16 workers we observe a saturation. When we looked at the total amount of CPU time used, Fig.2, we see the total CPU time is more or less constant up to 8 workers and above 16 workers. In the range of

[1] A Message Passing Interface, see http://www.mpi-forum.org/

[2] TFORM uses POSIX threads, or pthreads

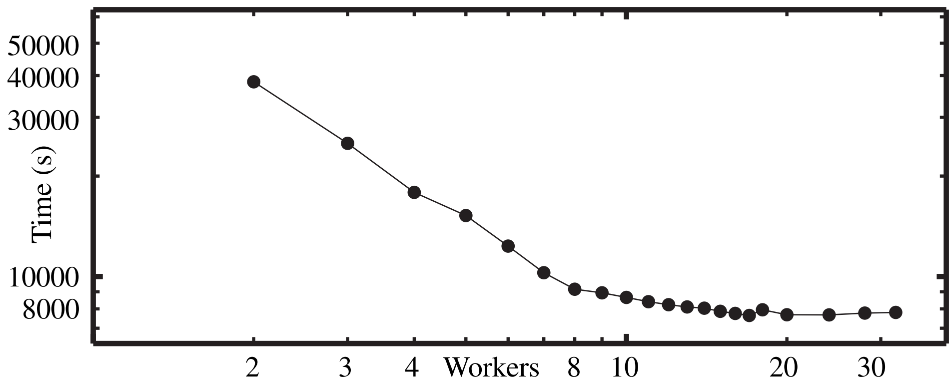

Figure 1: Running times of the Multiple Zeta Value TFORM program. The runs were for weight 23, up to depth 7.

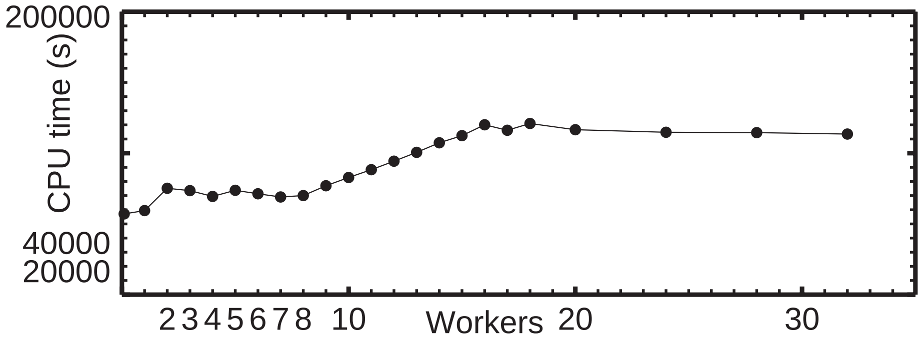

Figure 2: Total CPU time of the Multiple Zeta Value TFORM program.

8-16 workers however it increases steadily. This is responsible for the slower decline in real time in the first graph, because the pseudo efficiency (total CPU time divided by real time and divided by number of workers) remains more or less the same in this range. This is behaviour that is typical for hyperthreading. The total amount of work that can be obtained from this computer is about 9.5 times the amount that can be obtained from a single core.

The analysis of the data reveals also that TFORM needs about 20% overhead for the Multiple Zeta Program. This is more than for programs like Mincer. This may be due to the use of brackets from the master expression which may involve copious use of locks. This is still not completely clear though. The result is that for 8 workers the pseudo speedup (total CPU time divided by realtime) is 7.63 while the real speedup (compared to the FORM run) is 6.22. Of course, this is still very good. The maximum improvement we obtained was 7.45 for a run with 17 workers.

4. Recent development

Over the past years parallel FORM versions have picked up a number of new features:

- **Dollar variables**. By default, both ParFORM and TFORM switch into the sequential mode for each module which gives dollar variables a value during execution. But there are common cases when some dollar variables obtained from each term in each chunk can be processed in parallel in order to get a minimum value, a maximum, or a sum of results. Also, sometimes at the end of the processing of a term the value of the dollar variable is not important at all. Hence new module options have been implemented to help FORM to process these variables in parallel: `minimum`, `maximum`, `sum` and `local`.

- **Right-hand side expressions (RHS)**. This is not a problem for TFORM since all threads work with the same file system while it is a big problem for ParFORM since the expression may be situated in a scratch file but different nodes may have independent scratch file systems. For a long time ParFORM forced evaluation of modules with RHS expressions in sequential mode. Now ParFORM is able to perform RHS expressions in a real parallel mode.

- **InParallel statement**. A new statement was inplemented, `inparallel;`. This statement allows the execution of complete expressions in a single worker simultaneously. This is really useful when there are many short expressions, sometimes it gives a significant increase in efficiency.

In Fig. 3 we summarize the speedup curves for the TFORM running the MZV program on 8 CPU cores computer when various features are switched off/on. The legend is the following:

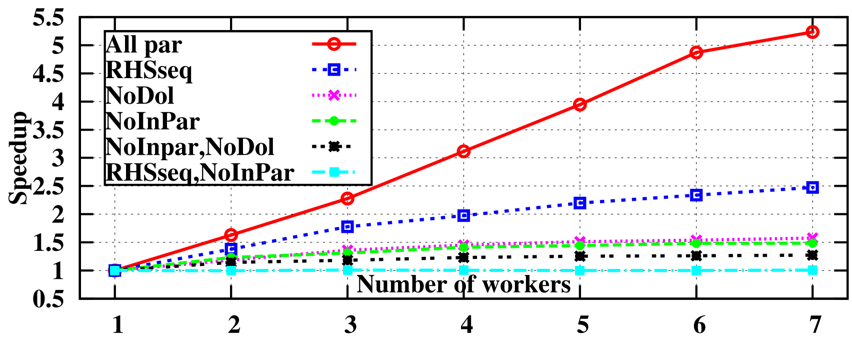

Figure 3: Results of the MZV program runs with various features switched off/on. The runs were for weight 20, up to depth 8.

- **All par** – all above mentioned features are implemented;
- **RHSseq** – modules with RHS expressions are forced into the sequential mode;
- **NoDol** – modules with dollar variables are forced into the sequential mode;
- **NoInar** – no InParallel statements;
- **NoInPar,NoDol** – modules with dollar variables are forced into the sequential mode, no InParallel statements;
- **RHSseq,NoInPar** modules with RHS expressions are forced into the sequential mode, no InParallel statements.

As we can see, all these new features are really important.

If FORM programs have to run for a long time the reliability of the hardware or of the software infrastructure becomes a critical issue. Program termination due to unforeseen failures may waste days or weeks of invested execution time. The checkpoint mechanism was introduced to protect long running FORM programs as good as possible from such accidental interruptions. With activated checkpoints FORM will save its internal state and data from time to time on the hard disk. This data then allows a recovery from a crash. The parallel FORM versions support this mechanism as well.

By default, data are saved at the end of each module. Usually this is too expensive. Optionally, the data may be saved only after some time interval. The scalability for ParFORM running BAICER N=16 for different intervals between checkpoints is depicted in Fig. 4. As one can see, even very

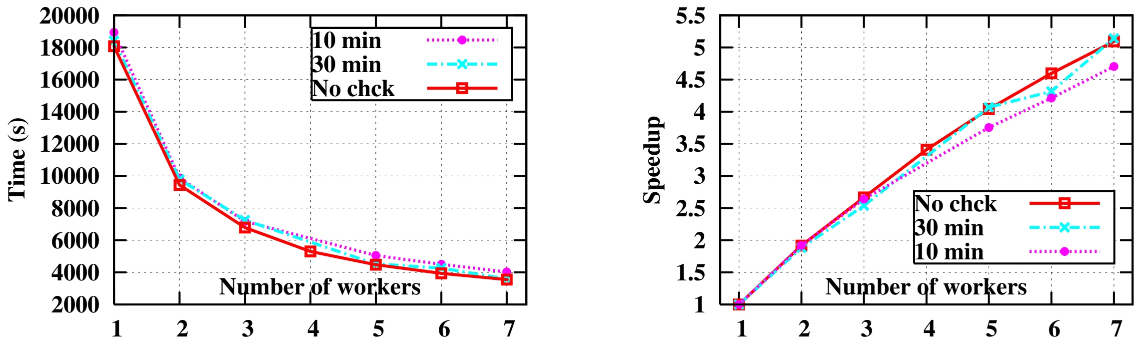

Figure 4: Absolute time and speedup curves for the test program BAICER without checkpoint mechanism (“NoChck”), checkpoints every 30 minutes (“30 min”) and every 10 minutes(“10 min”).

frequent checkpoints do not affect performance much.

Acknowledgments. This work was supported in part by DFG through SBF/TR 9 and by the FOM foundation.

References

[1] J. A. M. Vermaseren, [arXiv:math-ph/0010025];
J. A. M. Vermaseren, Nucl. Phys. Proc. Suppl. **183** (2008) 19, [arXiv:0806.4080 [hep-ph]].

[2] S. Moch, J. A. M. Vermaseren and A. Vogt, Nucl. Phys. B **688** (2004) 101;
A. Vogt, S. Moch and J. A. M. Vermaseren, Nucl. Phys. B **691** (2004) 129;
J. Blumlein and J. A. M. Vermaseren, Phys. Lett. B **606** (2005) 130;
Y. Schröder and A. Vuorinen, JHEP **0506** (2005) 051;
J. A. M. Vermaseren, A. Vogt and S. Moch, Nucl. Phys. B **724** (2005) 3;
R. Bonciani and A. Ferroglia, Phys. Rev. D **72** (2005) 056004;
Y. Schröder and M. Steinhauser, JHEP **0601** (2006) 051;
K. G. Chetyrkin, J. H. Kuhn and C. Sturm, Nucl. Phys. B **744** (2006) 121;
T. Aoyama, M. Hayakawa, T. Kinoshita and M. Nio, Nucl.Phys. B **740** (2006) 138.

[3] A. Retey and J.A.M. Vermaseren, Nucl. Phys. **B604** (2001) 281;
P.A. Baikov, K.G. Chetyrkin and J.H. Kuhn, Phys. Rev. Lett. **88** (2002) 012001;

P.A. Baikov, K.G. Chetyrkin and J.H. Kuhn, Phys. Lett. **B559** (2003) 245;
P.A. Baikov, K.G. Chetyrkin and J.H. Kuhn, Phys. Rev. **D67** (2003) 074026;
P.A. Baikov, K.G. Chetyrkin and J.H. Kuhn, Eur. Phys. J. **C33** (2004) 650;
S. Bekavac, hep-ph/0505174;
P. A. Baikov, K. G. Chetyrkin and J. H. Kuhn, Phys. Rev. Lett. **95** (2005) 012003;
P. A. Baikov, K. G. Chetyrkin and J. H. Kuhn, Phys. Rev. Lett. **96** (2006) 012003;
P. A. Baikov, K. G. Chetyrkin and J. H. Kuhn, Phys. Rev. Lett. **101** (2008) 012002;
A. Kotikov, J.H. Kuhn and O. Veretin, Nucl. Phys. **B788** (2008) 47;
P. A. Baikov, K. G. Chetyrkin and J. H. Kuhn, Phys. Rev. Lett. **104** (2010) 132004.

[4] D. Fliegner *et al*, [arXiv:hep-ph/9906426];
D. Fliegner *et al*, [arXiv:hep-ph/0007221]. M. Tentyukov *et al*, “Parallel Version of the Symbolic Manipulation Program FORM”, in: V.G. Ganzha *et al* (Eds.), Proceedings of the CASC 2004, Technische Universität München, Garching, Germany; [arXiv:cs.SC/0407066];
M. Tentyukov *et al*, Nucl. Instrum. Meth. A **559** (2006) 2248.

[5] M. Tentyukov and J.A.M.Vermaseren, PoS (ACAT08) 119.

[6] M. Tentyukov and J.A.M. Vermaseren, doi:10.1016/j.cpc.2010.04.009, [arXiv:hep-ph/0702279].

[7] J. Blumlein, D.J. Broadhurst and J.A.M. Vermaseren, [arXiv:0907.2557].