PROCEEDINGS OF SCIENCE

Current status of FORM parallelization

M. Tentyukov *†

Institut für Theoretische Teilchenphysik, Universität Karlsruhe, Germany

J.A.M. Vermaseren
Nikhef
Science Park 105
1098 XG, Amsterdam

We report on the status of the current development in parallelization of the symbolic manipulation system FORM. There are two parallel versions available, 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 of existing FORM programs will be able to take advantage of the parallel execution, without the need for modifications. In some cases some minor additions may be needed.

More information about FORM can be found on the FORM web page

`http://www.nikhef.nl/~form/`

XII Advanced Computing and Analysis Techniques in Physics Research
November 3-7 2008
Erice, Italy

*Speaker.
†Supported by SFB-TR9

1. Introduction

The symbolic manipulation system FORM [1] has been available for more than 20 years. It is specialized to handle very large algebraic expressions of billions of terms in an efficient and reliable way. That is why 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. None of the more spectacular calculations of refs [2, 3], would 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.

Some internal specifics [4, 5] allow FORM to deal with expression which are much large than the memory (RAM) available. The only restriction for the size of the expressions is the disk space which is rather cheap now. This means, the complexity of the problem solvable by FORM is restricted practically only by the time. In this context an improvement of efficiency is very important.

Parallelization is one of the most efficient ways to increase performance, so the idea to parallelize FORM is quite natural.

In the late nineties a joint project was started with the University of Karlsruhe for adapting FORM to run on parallel computers and clusters. This has led to the program ParFORM which uses most of the FORM sources with some extra code in addition. It has been described in the literature [4, 6] and several calculations have greatly benefited from it [3].

Last few years most leading vendors switch to dual- and multicore processors. Hence it was judged important to create a version of FORM that can make efficient use of such systems. The above considerations have led to the creation of TFORM [7], the multithreaded version of FORM.

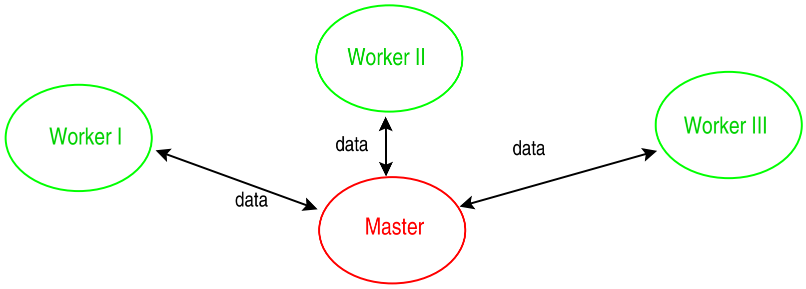
2. General concepts

FORM provides a special programming language adapted for the manipulating of large sequences of algebraic terms and the user supplies programs written in this language. The FORM program contains several parts called *modules*. The programs are executed module by module, see e.g. [4].

Mostly FORM allows local operations on single terms, like replacing parts of a term or multiplying something to it; non-local operations like replacing a sum of two terms by another term are not allowed. We refer to this property as the *locality principle*: all *explicit* algebraic operations are local. Non-local operations are allowed only implicitly in the sorting procedure at the end of the modules, when equivalent terms are summed up, and in some special references to the contents of brackets in the input expression.

All the existing approaches to the FORM parallelization consist of a “Master” which splits the incoming expression into chunks and distributes these chunks among “Workers”, see Fig. 1. The workers generate terms, sort them and send sorted terms back to the master. The master performs final sorting [4].

This is done absolutely “transparently” for the FORM programmer, no special efforts for parallel programming are required. The same FORM program runs in parallel.



$$\texttt{Local expr} = \underbrace{\texttt{a\^{}2+a*x}}_{\textit{To Worker I}} \underbrace{\texttt{+x\^{}3-b*2}}_{\textit{To Worker II}} + \underbrace{\ldots}_{\textit{To next workers}}$$

Figure 1: General parallelization concept

3. Models in use

There are two implementation of the above mentioned concept, ParFORM and TFORM, see Fig. 2. The workers are started only once at the startup.

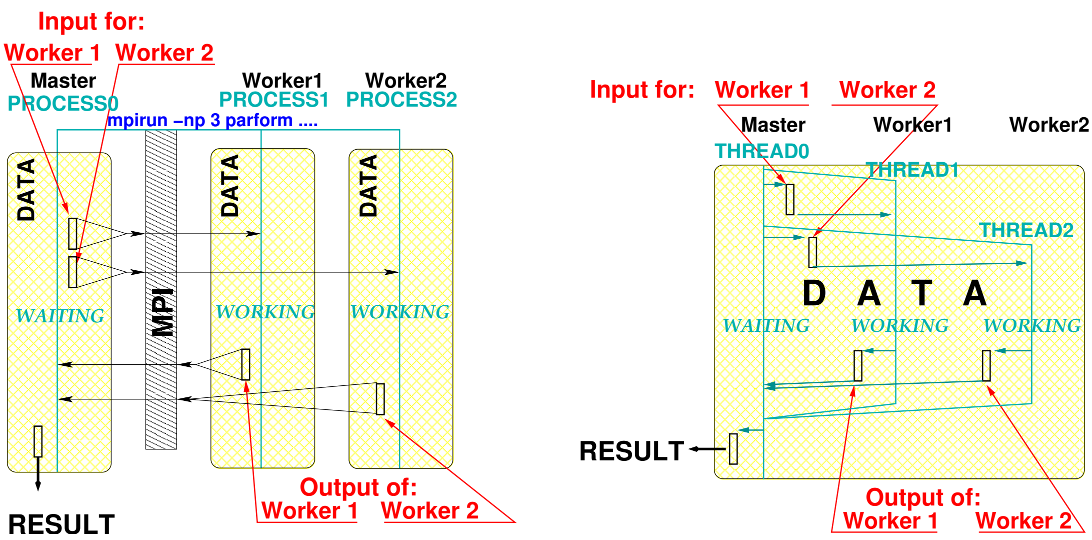


Figure 2: Structures of ParFORM and TFORM

ParFORM uses independent processes communicating via a message passing protocol named MPI[1] which makes it suitable to parallelize the FORM program not only on Symmetric Multi Processor (SMP) computers but also on clusters and Massively Parallel Processors[2]

TFORM starts a pool of POSIX[3] threads. A thread is a semi-process, that has its own stack, and executes a given piece of code. Unlike a real process, the thread normally shares its memory

[1] More about it can be found at `http://www.mpi-forum.org/`

[2] See, e.g.,`http://sawaal.ibibo.com/computers-and-technology/what-massively-parallel-processing-mpp-516077.html`.

[3] POSIX, "a Portable Operating System Interface for uniX" is the collective name for a family of related standards.

with other threads. If the computer is an SMP computer, i.e., it has several CPU cores, each worker can run on its own core thus speedup the process.

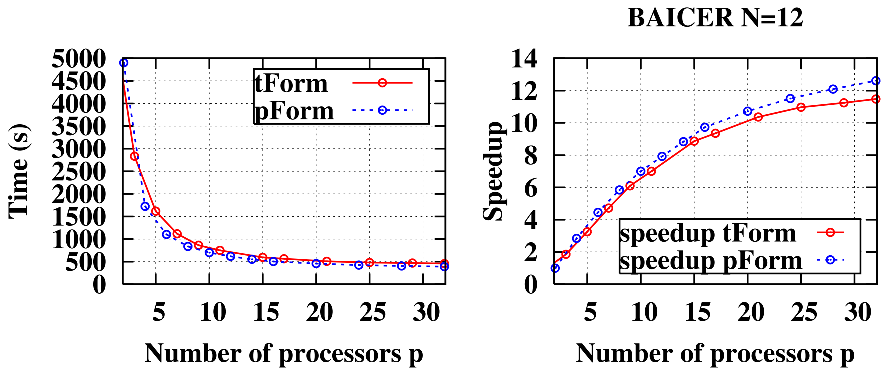


Figure 3: Computing time and speedup for the test program BAICER on the SGI Altix 3700 SMP server with 32x Itanium2 processors (1.3 GHz).

Both models work well and give a speedup of more than 10 on a 32-core computer, see Fig. 3. The speedup is almost linear up to 16 CPU cores.

4. Features and problems

One of the main ParFORM disadvantage is its dependence on MPI. Despite MPI being the industrial standard, various implementations are not binary compatible so ParFORM should be compiled with exactly the same MPI version which is installed on the computer. In contrast, TFORM requires no installation since it is just an executable file.

TFORM shares all resources of a single computer which sometimes leads to a competition between threads. Common bottlenecks are various buses, especially the front-side bus, and the local disk storage. In contrast, ParFORM is able to run independent processes on individual nodes which (theoretically) increases its scalability. On the other hand, shared address space allows TFORM to implement some features which are hardly possible for ParFORM. In the future we are planning to join these concepts in order to obtain the advantages of both of them, see Section 5.

Both ParFORM and TFORM have efficient techniques for load balancing: the Master immediately sends the next chunk to a ready worker. The smaller chunk the better load balancing but the worse overhead. With big chunks, it is easy to run into clusters of “bad” terms. A “bad” term produces a lot of new terms, many more than other terms. This means that in the end all other workers will be waiting for this worker to finish this or these term(s).

TFORM solves this problem by “stealing” the tail of the chunk which is processed by this “last” worker and re-distributing it among free workers. This is on by default; it can be switched on or off with the statements “`on ThreadLoadBalancing;`” and “`off ThreadLoadBalancing;`”. For ParFORM such kind of load balancing is impossible.

There are some variables in FORM that are an intermediate between local and global variables, so-called dollar-variables. Since they might be re-defined during processing of each term, their value can be nondeterministic in a parallel environment. Indeed, one worker could define such a variable and then the next one could overwrite this value before the first worker has used it. Hence their administration needs special attention.

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 natively can be processed 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`. Since TFORM is able to perform centralized administration of the shared objects the implementation of these options is rather efficient while ParFORM has to broadcast all dollar variables to all the workers and then collect them at the end of the module.

The next problem is Right-Hand Side (RHS) expressions, e.g.

```
L   F = a + b;
L   G = x + F;
```

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 and now it is able to treat them in parallel but less efficient then TFORM.

5. Outlooks and conclusion

TFORM is optimal on a relatively small number of CPUs since it does not suffer from a MPI overhead and can administrate various shared objects rather easily. But it is restricted to a single SMP computer. In the future we are planning to join the ParFORM independent-processes concept as a “coarse-grained” structure for parallelization on a cluster and the TFORM thread-based approach as a “fine-grained” parallelization on multi-core cluster nodes, see Fig. 4.

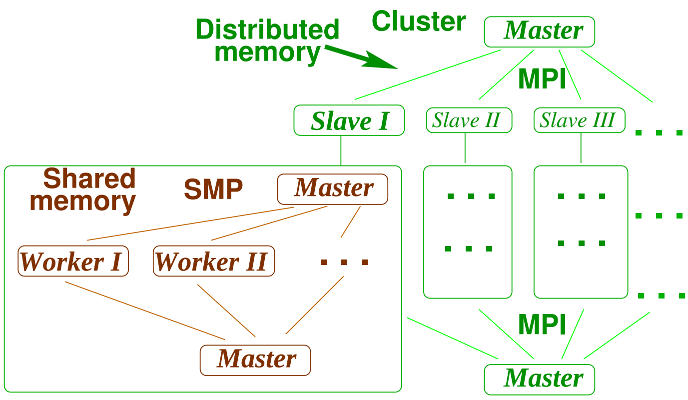


Figure 4: Combination of ParFORM and TFORM

At the present moment,

- Both ParFORM and TFORM are able to execute almost all FORM programs in parallel.
- ParFORM supports more hardware architectures. TFORM supports parallelization of more FORM features.
- ParFORM requires MPI, TFORM doesn't, which makes it much easy to deploy.
- TFORM is optimal for parallelization on a small (≤ 8) number of CPUs. ParFORM is optimal for parallelization on a large (≥ 6) number of CPUs.

In order to get benefits from a parallel version of FORM, some specific hardware is required.

ParFORM can be used on clusters with Gigabit Ethernet interconnection but it is better to have a faster network, like InfiniBand with non-blocking topology. Both the parallel FORM versions need a huge and fast parallel disk storage system. For example, it is not enough for ParFORM to have a cluster with a Fast Ethernet NFS disk system. It is important to have huge fast local disk storages on each of the nodes (hundreds GB per CPU core).

References

[1] J. A. M. Vermaseren, arXiv:math-ph/0010025.

[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;

[4] 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, J.A.M. Vermaseren, Comput. Phys. Commun. **176** (2007) 385.

[6] D. Fliegner *et al*, arXiv:hep-ph/9906426;
D. Fliegner *et al*, arXiv:hep-ph/0007221.

[7] M. Tentyukov and J.A.M. Vermaseren arXiv:hep-ph/0702279