TTP99-14
September 1998

The Parallelization of the Symbolic Manipulation Program FORM *

D.Fliegner†, A.Rétey†, J.A.M.Vermaseren‡

†*Institut für Theoretische Teilchenphysik, Universität Karlsruhe, D-76128 Karlsruhe, Germany*
‡*NIKHEF, P.O. Box 41882, 1009 DB, Amsterdam, The Netherlands*

Abstract. After a short introduction to the mechanisms behind the sequential version of the symbolic manipulation program FORM we report on the status of the project of its parallelization on a variety of computer architectures and show first results, which indicate a significant speedup.

I INTRODUCTION

In the recent years the symbolic manipulation program FORM [1] has been used in many calculations in high order perturbation theory [2–4] because of its ability to deal with huge symbolic expressions. Often the algorithms used are well known, but despite of the development of faster computers the limiting factor in those calculations is the execution time of the programs, which in some cases reaches the order of months. To speed things up the obvious thing to do is to parallelize FORM. In the following we will have a look into the mechanism FORM uses and explain why it is well suited for a parallelization. We will report on the further optimization of FORM on certain architectures and eventually present a version of FORM that sorts in parallel.

II SYMBOLIC MANIPULATION WITH FORM

We cannot possibly present the details and will concentrate on the basic ideas and the relevant points for parallelization. FORM is a symbolic manipulation program that—unless other programs—has a rather small instruction set in order to keep things as simple and reliable as possible. With few exceptions all data manipulations are done locally (on single terms). This has a clear influence on the speed of the program and makes it possible to use hard disks in an efficient

* talk given by D.Fliegner at CHEP'98, Chicago, Aug 31 - Sep 4 1998, published in Proceedings of CHEP'98 (http://www.hep.net/chep98/PDF/Sessions.html).

way if the problem size is bigger than the actual memory. FORM has a variety of data types. In fact part of its speed is due to its strong type orientation (most of the properties of the objects are known from the beginning). Some of the data types are very useful in high energy physics (like vectors, indices, dotproducts and γ-matrices).

As is indicated in the data flow diagrams (figures 1 and 2)

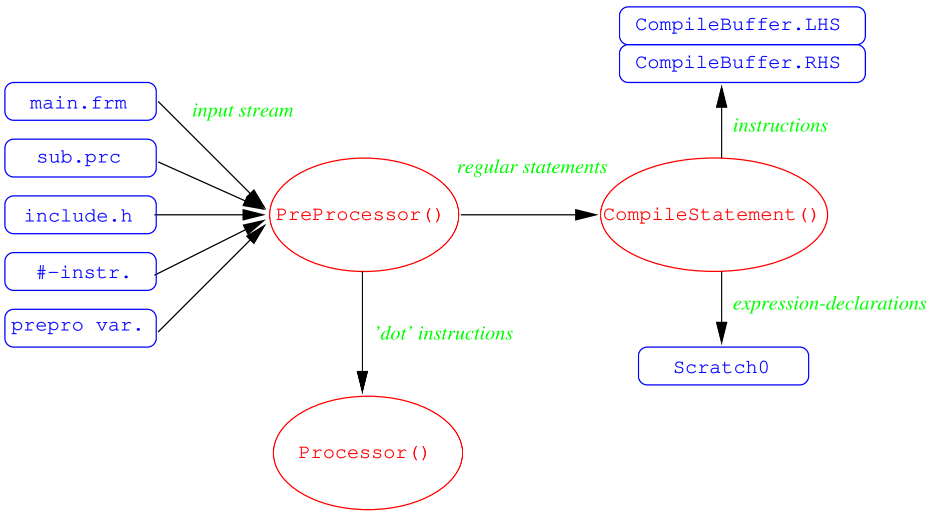

FIGURE 1.

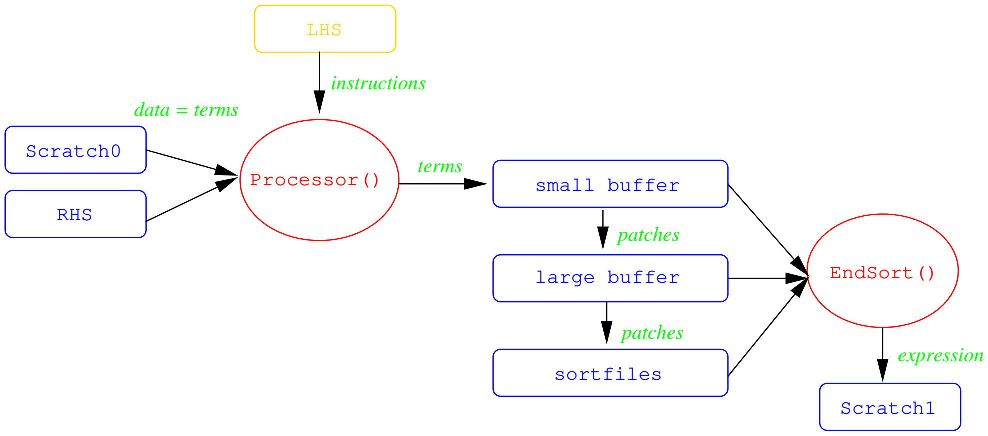

FIGURE 2.

the program basically consists of three parts namely the preprocessor, the compiler and the processor. The preprocessor reads the input from different streams and produces the actual program text, whereas the compiler translates the ASCII input text into the internal representation. If a dot-instruction (denoting the end of a module) is found, the actual work, the generating and sorting of terms, is done by the processor. All statements in a module are evaluated term by term resulting in a tree of generated and a tree of sorted terms (see figure 3)

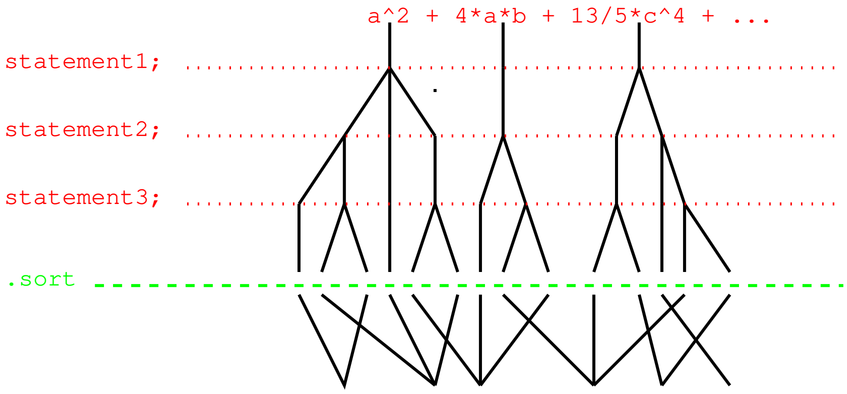

FIGURE 3. A tree of generated and sorted terms.

The sorting in FORM is performed in several steps (see figure 4). First the

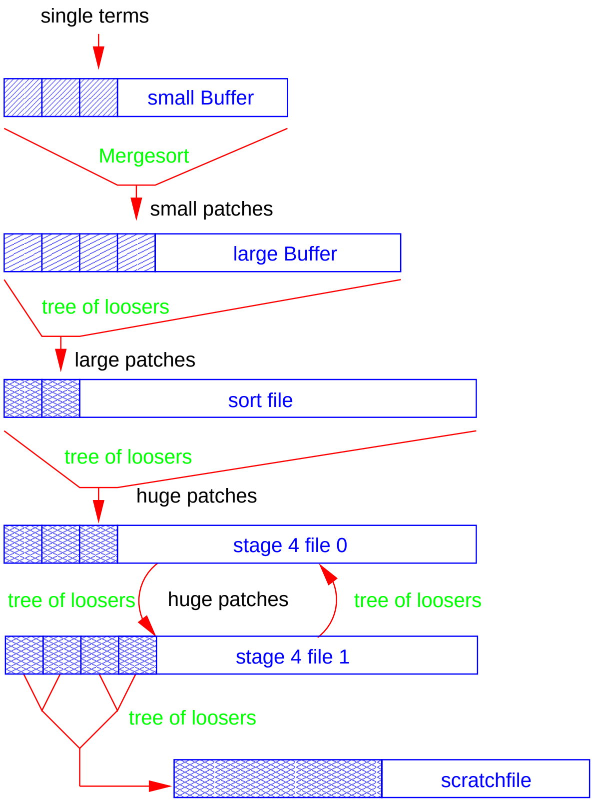

FIGURE 4. FORM's sorting mechanism.

single terms are written into the so called small buffer. If the small buffer is full, the terms are sorted (using the merge sort algorithm) into the large buffer. If the large buffer is full the patches are sorted into a sort file on the disk (using the tree of loosers algorithm [5]). If that sort file is still not big enough to hold the terms, the terms are sorted entirely on disk using two sort files. The whole procedure can be controlled by adjusting the size of the buffers in a set file that is read by FORM in the beginning. The problem size is limited by the disk space, not by the memory size.

III OPTIMIZING FORM

Before getting started with the parallelization it is of course a good idea to find optimizations that can be done easily. For instance a close look into the DEC C compilers options resulted in a speedup of a factor of 1.7 on DEC Alpha workstations. Internally terms are represented as a series of words, usually (on 32 Bit machines) of 16 Bit length. For 64 Bit machines (like the DEC Alpha workstations) a 32 Bit word is a much better choice, yielding an additional speedup of a factor of 1.8, albeit at the cost of doubling the data volume.

IV PARALLELIZING THE SORTING IN FORM

For the the parallelization it is of course essential to analyze the sequential version carefully. First a profiling had to be done in order to find the time consuming parts of the program. In typical FORM applications by far most of the time is spent in generating and sorting. The time needed for preprocessing and compiling is usually negligible. The tree structures FORM uses for generating and sorting terms and the fact that only local operations are performed result in a natural target for parallelization.

Another important point is the selection of hard– and software. Clearly it is a good idea to write a portable parallel program that will run on almost any computer and operating system. In our approach mainly the message passing libraries PVM [6] and MPI [7] are used. Different implementations are available for almost any UNIX clusters and there are special implementations for SMP machines using shared memory instead of the usual network protocols. In fact FORM can be compiled on the different architectures without major changes in the source code.

Up to now only the parallelization of the sorting has been completed, using the following mechanism: the terms are generated in a master FORM process and written into the small buffer like in the sequential version. If the small buffer is filled, it is sent to a slave FORM process (usually running on a different machine) and sorted there. Eventually the master process collects all sorted patches again and performs the final stages of the sorting. Of course this model suffers from an intrinsic saturation phenomenon (the slaves cannot sort faster than the master generates), a problem that will disappear once parallel generating of terms is implemented.

For the first test of the parallel sorting routines we used three entirely different machines:

- Digital Workstation 500/500 cluster running DEC UNIX 4.0B,
  8 processors, 500MHz, 512MB RAM and 4GB disk each,
  interconnection: Fast (100MBit/s) Ethernet, software: MPICH.
- IBM RS6000/SP running AIX 4.2.1,
  168 (in total 256) processors, 120MHz, 512MB,
  interconnection: special low latency switches, software: MPI.
- ALR Quad6 SMP machine running Solaris 2.6 (Intel),
  4 processors, 200MHz PentiumPro, 512MB RAM and 2 4GB disks
  interconnection: shared memory, software: MPICH (shared memory).

and the following simple program that essentially amounts to a pure sorting problem:

```
#ifndef `n'
#define n "2"
#endif

S a1,...,a`n';
S x;

L F = x^2;
id x = a1+...+a`n';

#if `n' > 3
id a1 = -a4-...-a`n';
#endif

print;
.end
```

The program computes a (huge) polynomial and then subtracts most of the terms again. The answer is of course known from the beginning ($a_2^2 + 2a_2a_3 + a_3^2$) and can be used as a check. The parameter n is used to adjust the problem size. On the DEC workstations $n = 1000$ corresponds to a problem size of 55MB, which fits into the memory easily, whereas $n = 5000$ corresponds to a problem size of 1359MB, which is larger than the memory of any of the single machines. The following plots (figures 5 and 6) show the results of parallel sorting for the two cases considered. The straight dotted lines represent the amount of time that is spent in the sequential part of the program, i.e. the preprocessing, compiling and generating. We get a considerable decrease in the wall clock time for all architectures, if the problem size is large enough. On the machines that have fast interconnections between the processes we still find an acceleration for small problem sizes, whereas the fast ethernet connections of the DEC Alpha cluster are obviously not sufficiently fast.

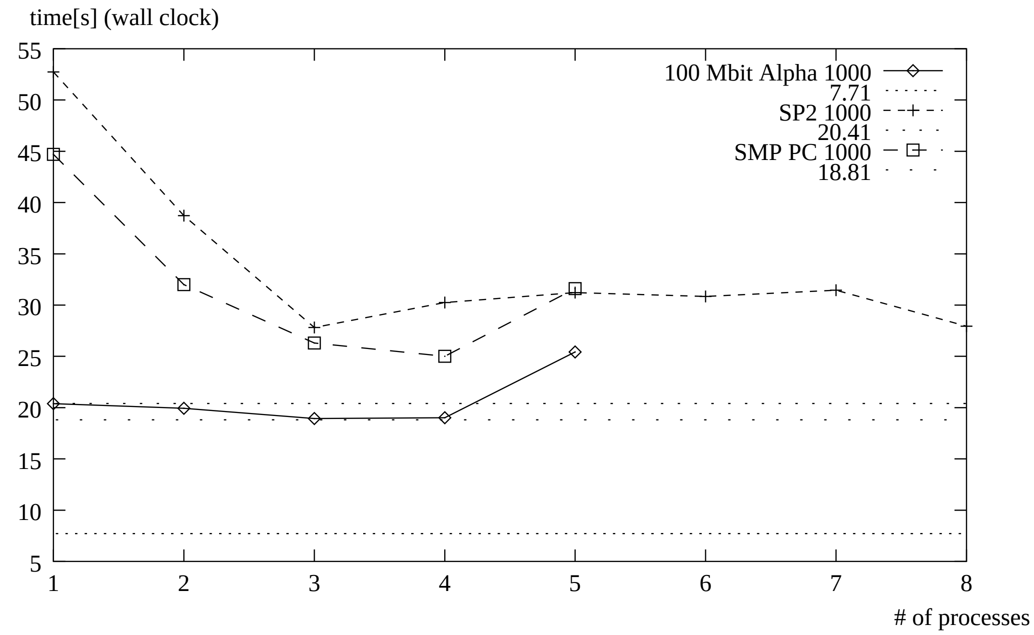

FIGURE 5. wall clock time, problem size $n = 1000$

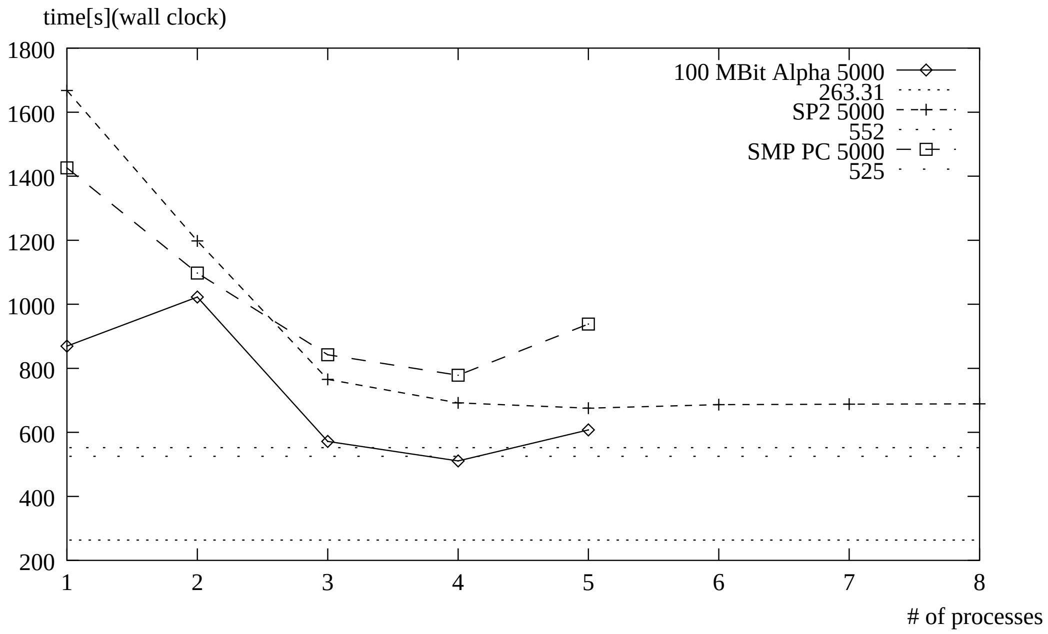

FIGURE 6. wall clock time, problem size $n = 5000$

Note that the wall clock time does not increase much for a large number of processes on the IBM SP. The increase for the DEC Alpha cluster is caused by the network overhead, for the SMP machine by the fact that only 4 processors are available.

In order to analyze the speedup in the sorting in detail we subtract the time spent in the sequential part and normalize the sequential value to 100 (see figures 7 and 8). In addition the theoretical speedup curve is shown.

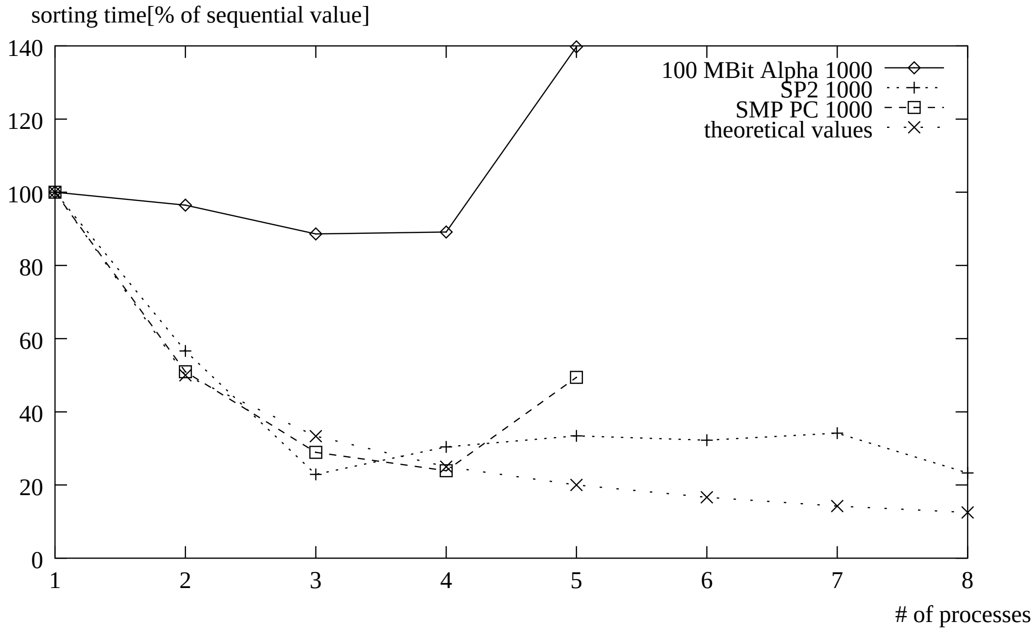

FIGURE 7. speedup in sorting, problem size $n = 1000$

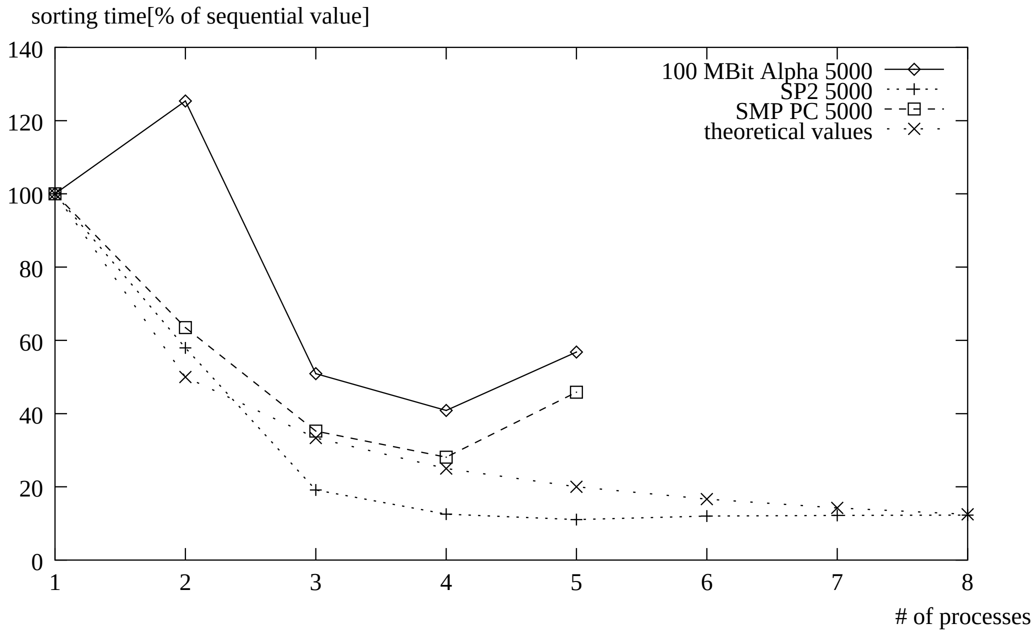

FIGURE 8. speedup in sorting, problem size $n = 5000$

For machines with fast process interconnections the theoretical values are reached. Large problems' speedup on the IBM SP is even bigger because of parallel disk use. Even on the DEC Alpha cluster the acceleration is considerable for large problem sizes. Of course, in absolute numbers the sequential version on a DEC Alpha machine is still almost as fast as the parallel version on the IBM SP.

V CONCLUSIONS AND OUTLOOK

We have shown that FORM is suitable for parallelization and discussed the implementation of parallel sorting using the standard message passing libraries MPI and PVM on a variety of architectures. The speedup in sorting reaches the theoretically expected value, if the problem size is large enough and the communication between the processes is sufficiently fast. At the moment a DEC Alpha cluster and a Linux PC cluster with faster interconnections based on a Myricom network [8] and special implementations of the message passing libraries that avoid protocol overhead [9] are under construction. The estimated bandwidth of this hard– and software combination is a factor of ten higher than the 100Mbit Ethernet, while the latency will be at least a factor of ten lower. We expect the speedup curves of this clusters to be very similar to the IBM SP. On the software side we are working on the parallel generating of terms. Also different options for reducing the data volume are under investigation, namely using hash tables for sorting and better compression routines. Because we are only using a small subset of routines of the messages passing libraries PVM and MPI we decided to implement our own message passing routines based on System V IPC. Besides the Pentium Pro based SMP machine a version for SUN and SGI SMP machines is under construction.

REFERENCES

1. J.A.M. Vermaseren, *Symbolic Manipulation with FORM*, published by CAN (Computer Algebra Nederland), Kruislaan 413, 1098 SJ Amsterdam, 1991, ISBN 90-74116-01-9.
2. S.A. Larin, T. van Ritbergen, J.A.M. Vermaseren, *Phys.Lett.* **B400** 379 (1997); S.A. Larin, P. Nogueira, T. van Ritbergen, J.A.M. Vermaseren, *Nucl. Phys.* **B492** 338 (1997); T. van Ritbergen, R.G. Stuart, preprint UM-TH-98-15, hep-ph/9808283.
3. S. Laporta, E. Remiddi, *Phys. Lett.* **B379** 283 (1996).
4. K.G. Chetyrkin, R. Harlander, J.H. Kühn, M. Steinhauser, *Nucl. Phys.* **B503** 339 (1997); K.G. Chetyrkin, R. Harlander, M. Steinhauser, *Phys. Rev.* **D58** 014012 (1998); R. Harlander, T. Seidensticker, M. Steinhauser, *Phys. Lett.* **B426** 125 (1998).
5. D.E. Knuth, *The Art of Computer Programming*, Addison Wesley 1997.
6. http://www.epm.ornl.gov/pvm/.
7. http://www.mcs.anl.gov/mpi/; http://www-c.mcs.anl.gov/mpi/mpich/.
8. http://www.myri.com.
9. http://wwwipd.ira.uka.de/ParaStation/PSM/.