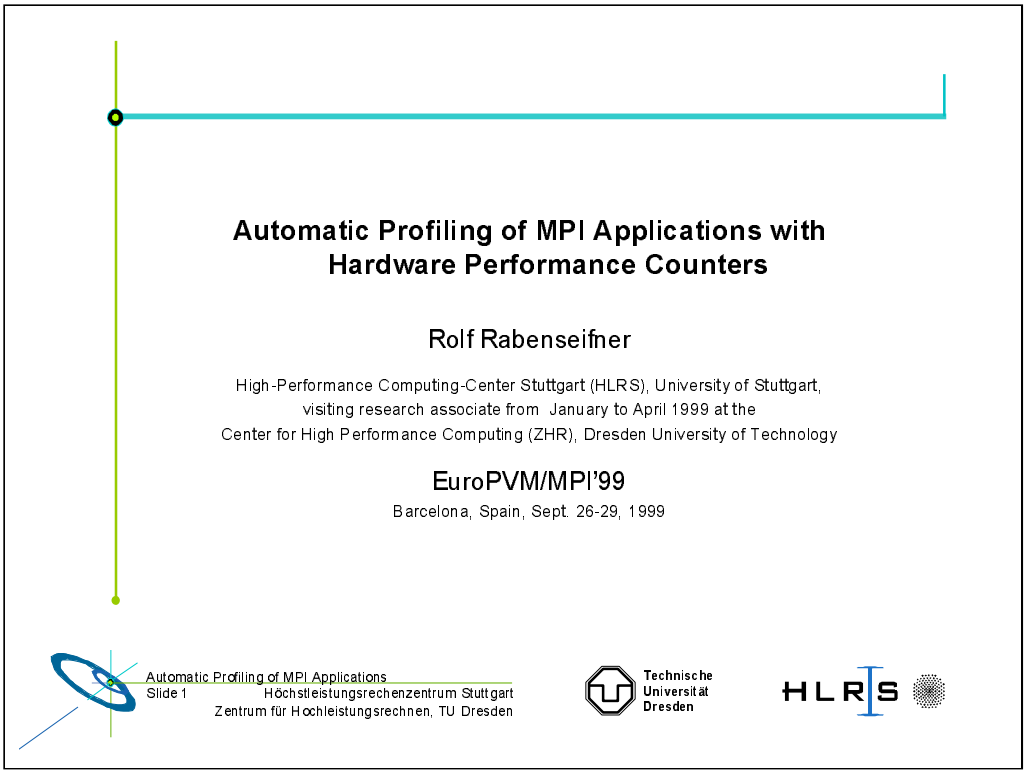


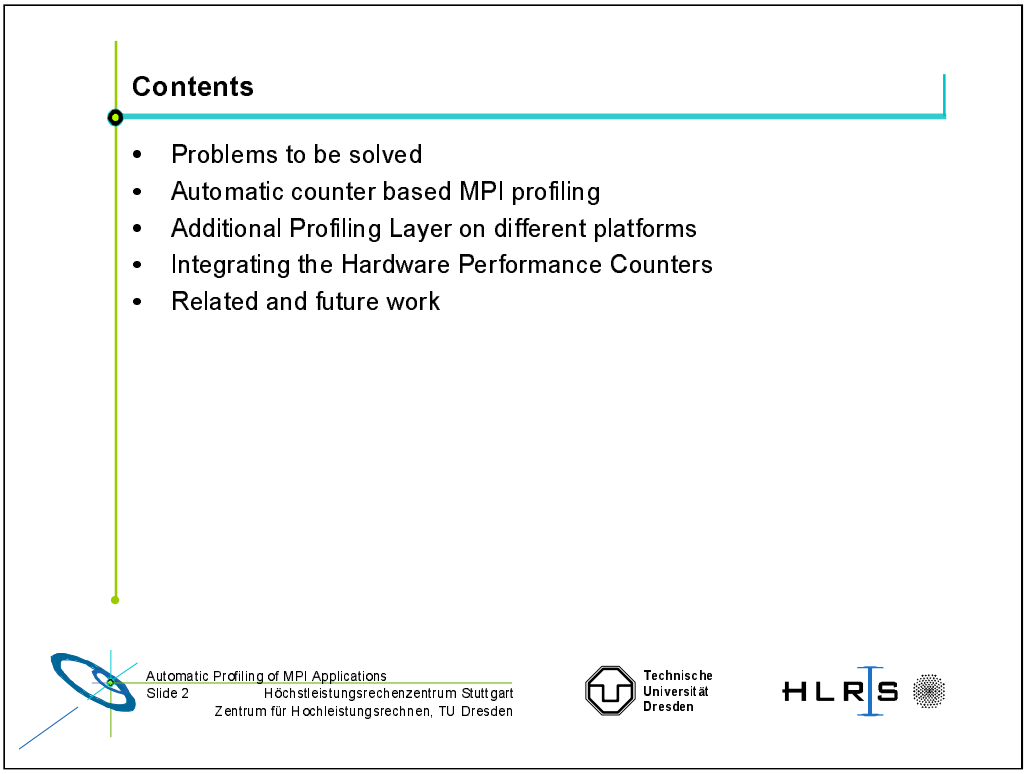
In the next minutes I shall introduce the automatic profiling of MPI applications and the integration of the hardware performance counters.

Additional information:

The author is responsible for message passing programming models at the High-Performance Computing-Center of the University of Stuttgart.

He is a member of the MPI-2 Forum.

Some of the work was done while he was a visiting research associate at the Center for High Performance Computing at the Dresden University of Technology.

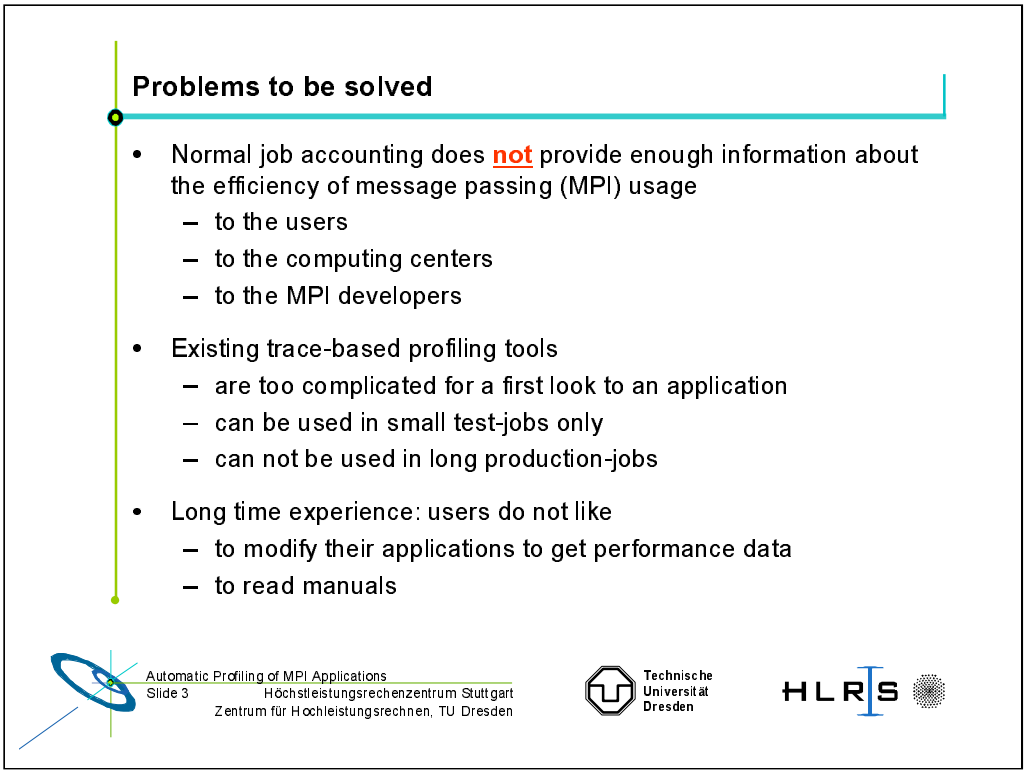


Here the outlook of my talk.

- After reporting the problems to be solved,
- I will give a brief overview on our automatic counter based MPI profiling.

In this talk I will focus on two aspects:

- The integration of an additional profiling layer into the default MPI library on different platforms, and
- I will show my strategy to integrate the hardware performance counters, which allow to profile not only the MPI communication, but also the computation.
- At the end I will show future plans and I shall give a summary.



Normal job accounting does not provide enough information about the efficiency of message passing usage

to the users

to the computing centers and

to the MPI developers.

Existing trace-based profiling tools

are too complicated for a first look to an application.

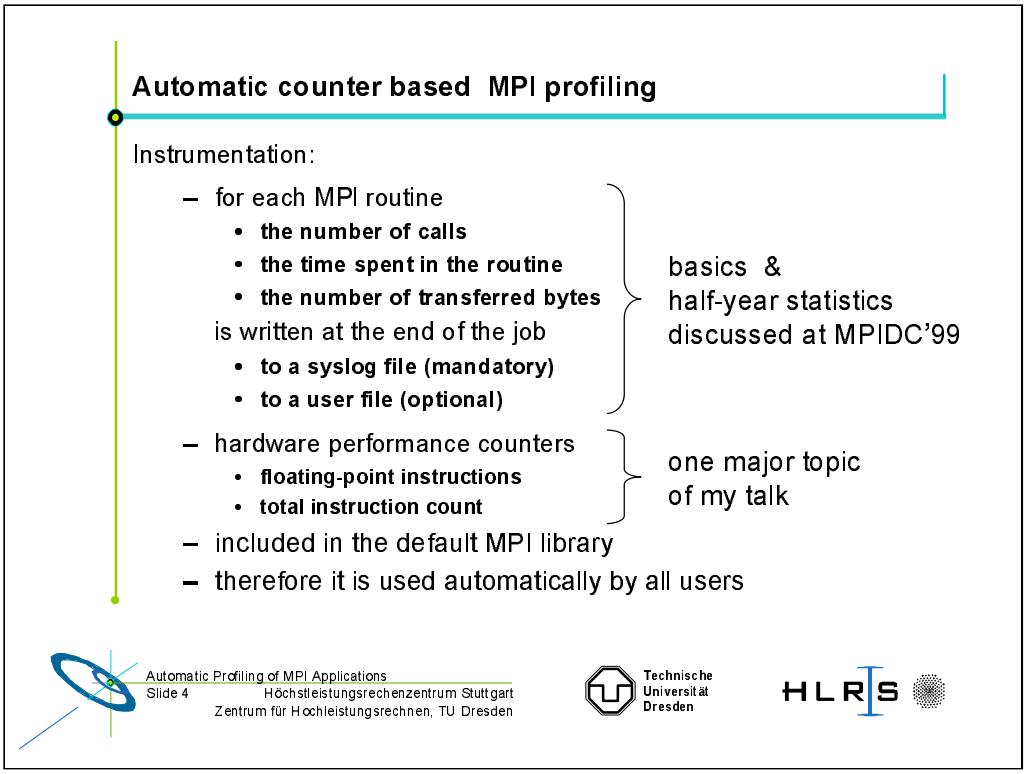
Such tools can be used in small test-jobs only

and they can not be used in long production-jobs.

Long time experience has shown, that the users do not like

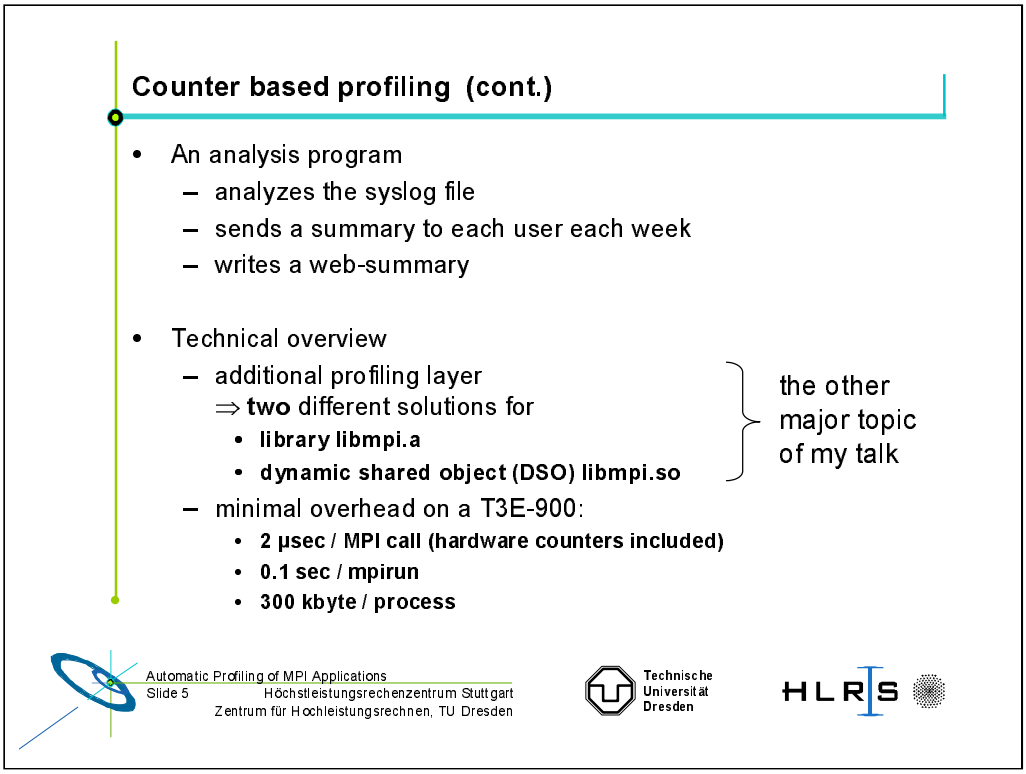
to modify their applications to get performance data

and they do not like to read manuals.



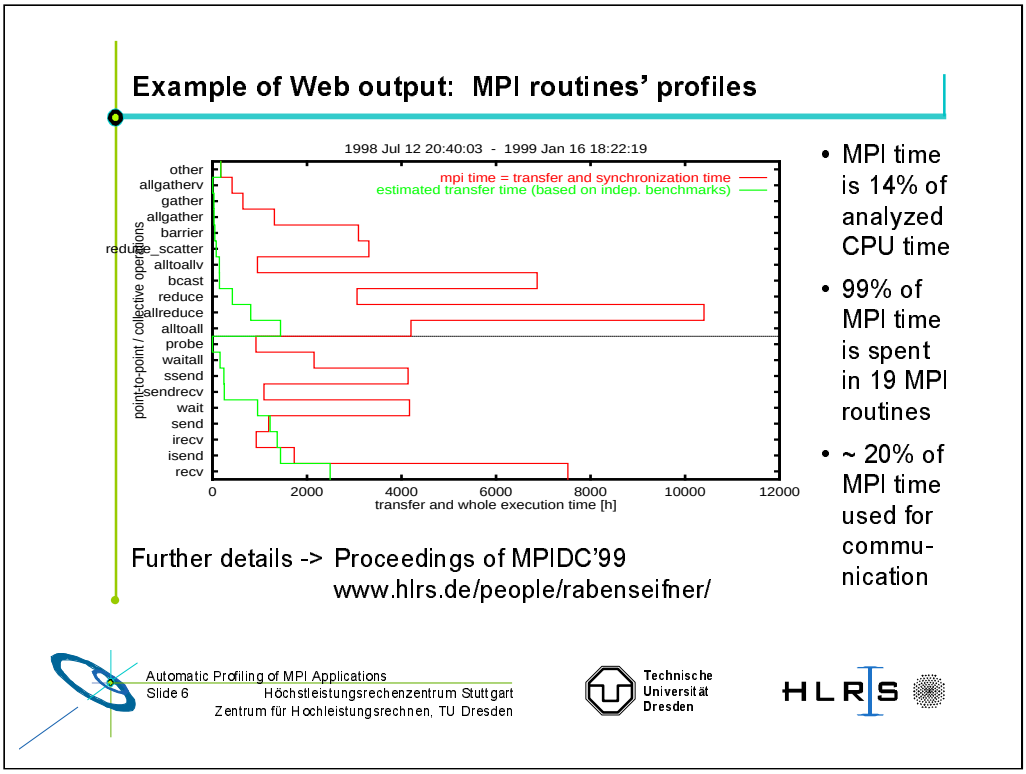
To solve these problems, our computing center has developed an automatic counter based MPI profiling.

- For each MPI routine
  - the number of calls,
  - the time spent in the routine, and
  - the number of transferred bytes
- is written at the end of the job
  - to a syslog file
  - and optionally to a user file.
- After integrating the hardware performance counters, now, we also profile
  - the floating-point instructions
  - and the total instruction count.
- The whole instrumentation is included in the default MPI library.
- Therefore it is used automatically by all users that use the default MPI library.
- Basics about the MPI profiling and a half-year statistics were presented at the MPI Developers Conference this year in Atlanta.
- The integration of the hardware performance counters is one of my major topics in this talk.



- I mentioned already, that all profiling information is written to a special syslog file.
  - This syslog file is analyzed with an analysis program each week.
  - The analysis program sends a summary to each user each week
  - and it writes a web-summary
- Two technical topics should be mentioned
  - To integrate this automatic counter profiling into the MPI library, it was necessary to include an additional profiling layer, without loosing the standardized profiling capabilities.
  - ⇒ For this problem, there are two different solutions:
    - one for Unix libraries
    - and another for dynamic shared objects.

    } This is the other major topic of my talk.
  - The MPI profiling can be used in each MPI job due to its minimal overhead. On a T3E-900, it is
    - 2 µsec for each MPI call
    - 0.1 sec for each mpirun to write the counters on the syslog file
    - and 300 kbytes memory overhead in each process.



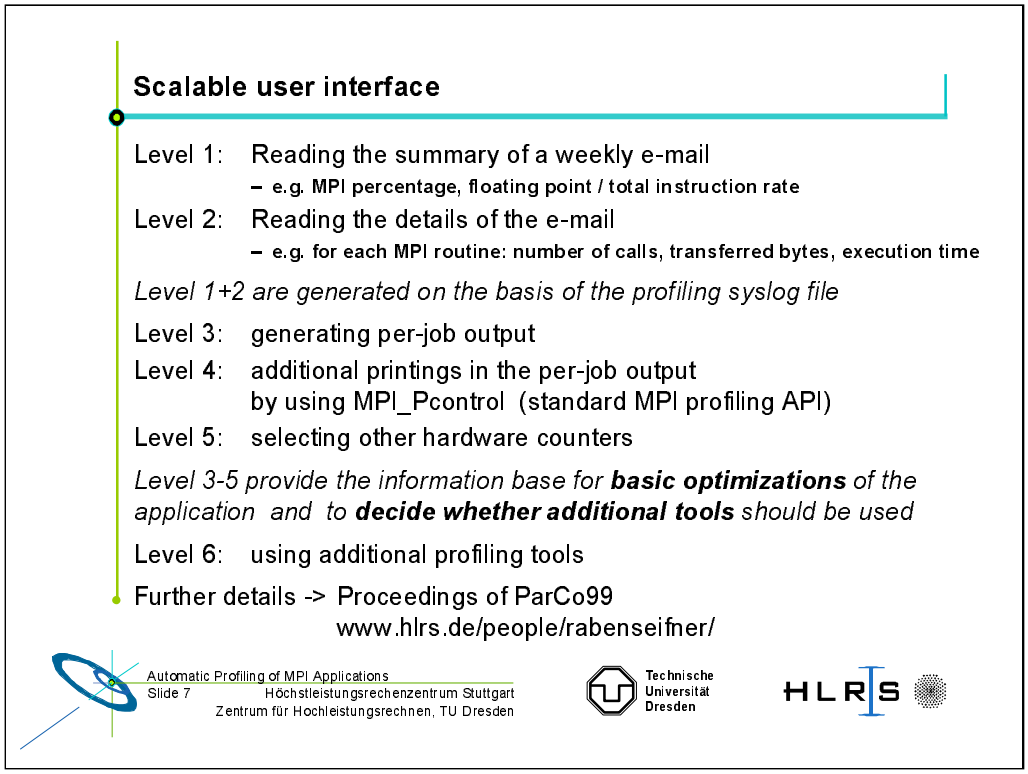
Before discussing the problems and solutions about the integration of an additional profiling layer and the integration of the hardware performance counters,
I want to show some results already published.

In a half-year statistics on our 512 processor T3E system, we could see, that about 14% of the application CPU time was spent in MPI routines.

99% of this MPI time was spent in 19 MPI routines.

The diagram plots with the red line the number of accumulated CPU hours spent in each of these 19 routines.

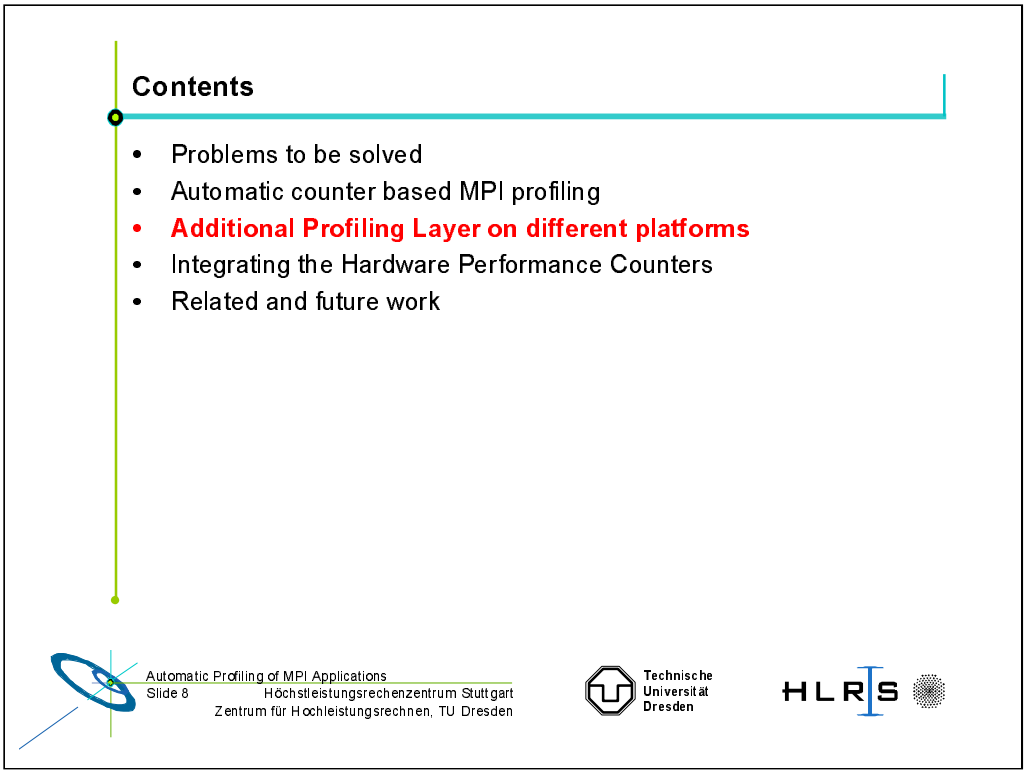
A rough estimate about the real transfer time -- here the green line -- has shown that only about 20 % of the MPI time is used for transferring data, the rest is blocking time.



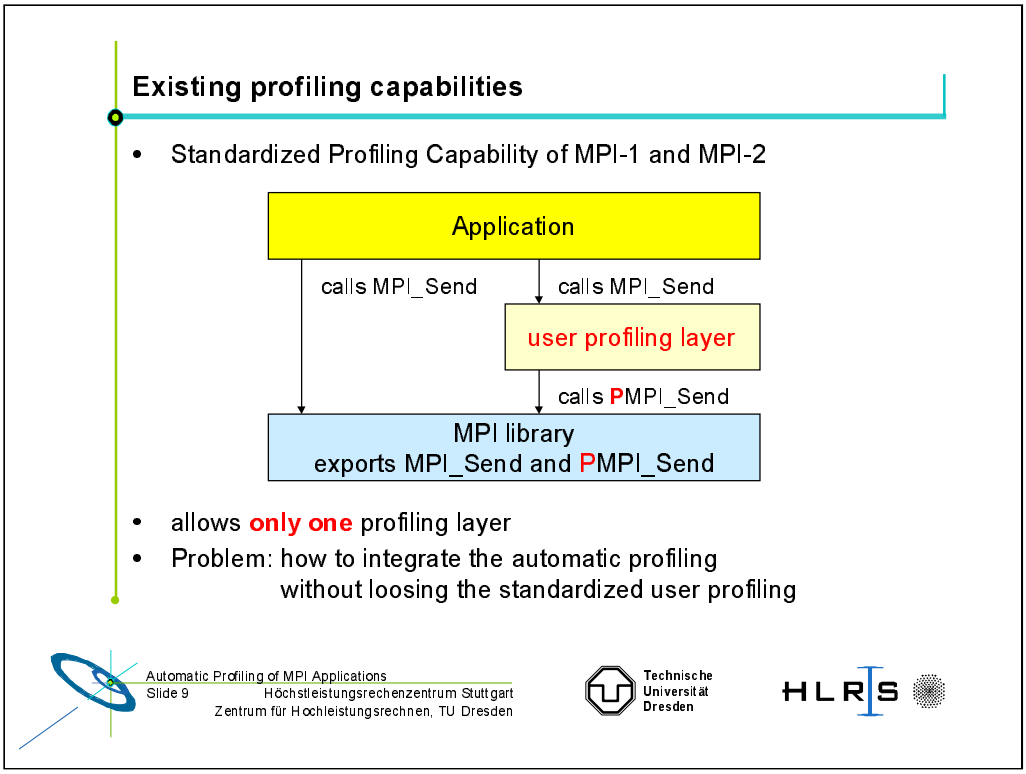
The automatic counter profiling has a Scalable user interface.
This means, that the user can choose between different levels to get the right amount of performance data with an adequate amount of work.

- On Level 1 the user has only to read the summary of a weekly e-mail.
  - He sees his MPI percentage, his floating-point instruction rate and his total instruction rate.
- On Level 2 he can read the details of the e-mail.
  - For each MPI routine, he gets the number of calls, the transferred bytes, the execution time, and a rough estimation of the transfer and synchronization part of the MPI time.
- *Level 1 and level 2 are generated on the basis of the profiling syslog file.*
- On Level 3 the user can generate the optional profiling output in each job.
- On Level 4 he can generate additional printings in the per-job output by using MPI_Pcontrol.
- On Level 5 he can select other hardware counters.
- *Level 3 to 5 provide the information base for basic optimizations of the application and to decide whether additional tools should be used.*
- And on Level 6, additional profiling tools can be used.

This means, that this automatic counter profiling closes the gap between the accounting information and the complex trace-based profiling tools, like VAMPIR.
Further details about the scalable user interface have been published at the ParCo in Delft.



After this overview we first look at the problem to integrate an additional profiling layer into the existing MPI libraries.



The MPI standard defines a standardized profiling capability.

Between the application, that calls MPI routines, and the MPI library, one can insert a profiling layer without modifying the application.

For example VAMPIR is such a profiling layer.

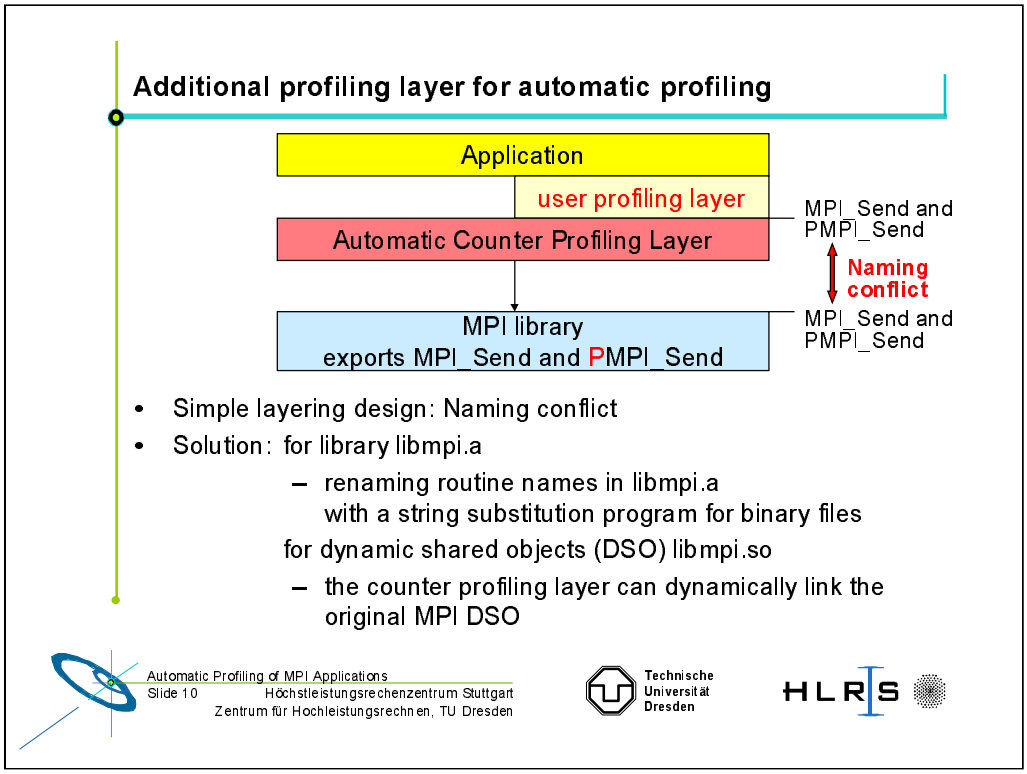
Technically, the profiling layer is a collection of wrapper routines.

The wrapper routines export some or all MPI routine interfaces and they import the **P**MPI-profiling interface.

The MPI library itself exports the MPI **and** the PMPI interface.

This trick can be used only for one profiling layer.

Our problem is, how to integrate the automatic profiling without loosing the standardized user profiling



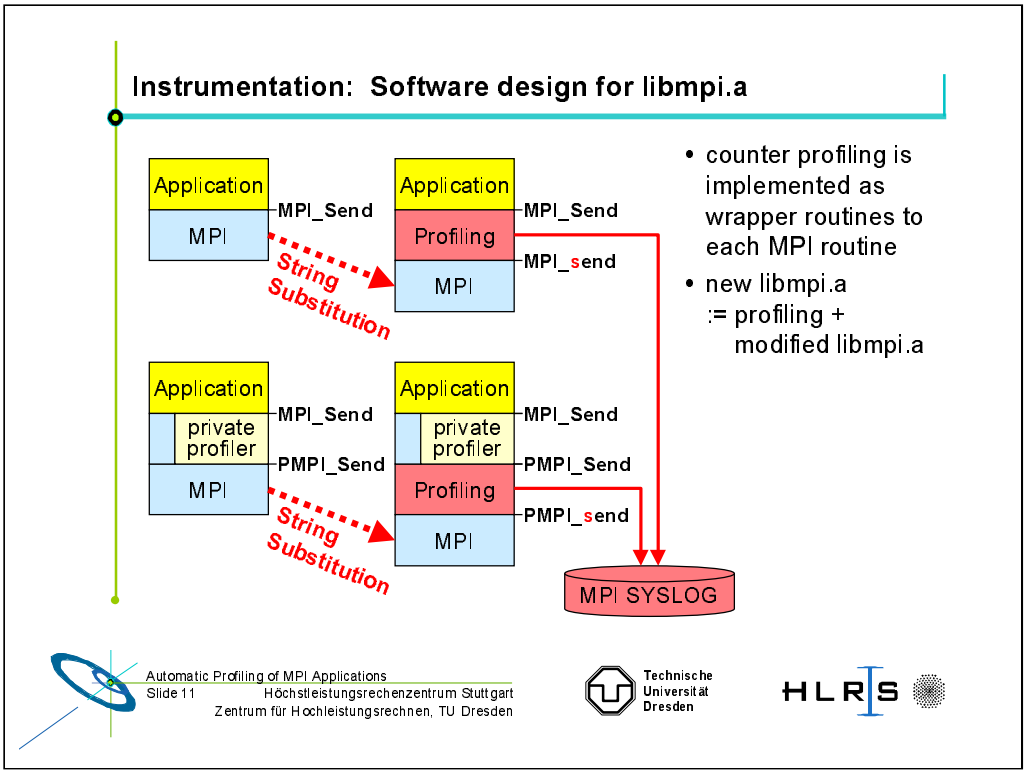
• With a simple layering design, we get naming conflicts.

The automatic counter profiling layer has to export the MPI and PMPI interface and has to import the same names from the vendor's MPI library.

• To solve this naming conflict, there exist two different solutions:

- For Unix libraries,
  one can rename the routine names in libmpi.a
  with a string substitution program for binary files.
- If the MPI library is implemented as a dynamic shared object,
  the counter profiling layer can dynamically link the original MPI DSO.

The following slides show these two different software designs.



The counter profiling instrumentation is implemented as wrapper routines to each MPI routine.

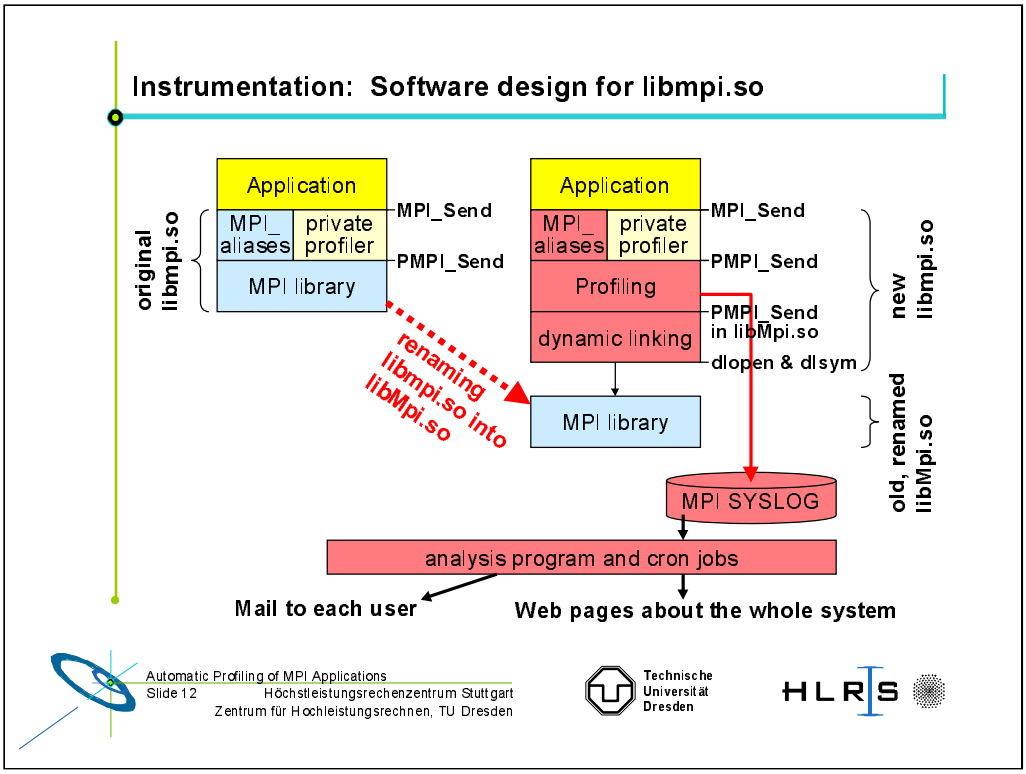
If the vendors MPI library is implemented as a Unix library, then with this software design, we can include the counter profiling into the MPI library:

In the original MPI library, we substitute the MPI routines' names.

The uppercase letter of the name -- here the S of MPI_Send -- is changed to a lowercase letter.

The new libmpi.a consist of the profiling layer and the modified old MPI library.

The lower picture shows the same as the upper part, but with standardized profiling capabilities included.



Another design is necessary if the MPI library is implemented as a dynamic shared object.

Here, only the original MPI library must be renamed.

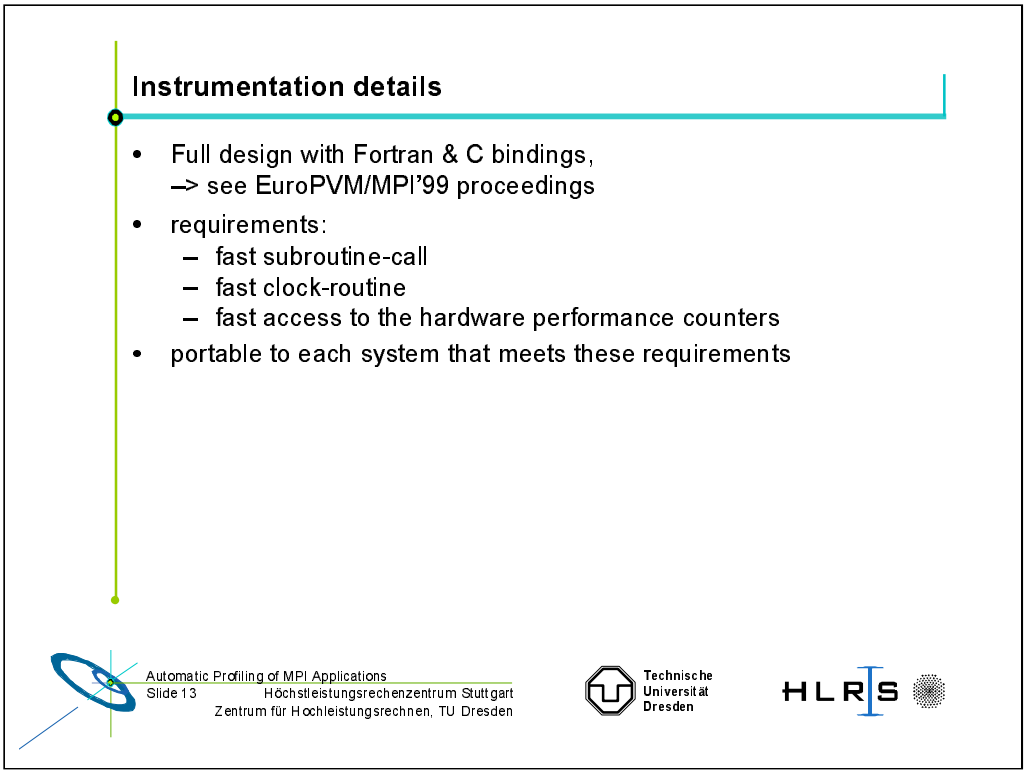
The new MPI library consist only of our counter profiling layer.

The profiling layer dynamically links at runtime the original renamed library.

On this picture we also see the analysis program.

Each weekend it reads the syslog file and sends a mail to each user.

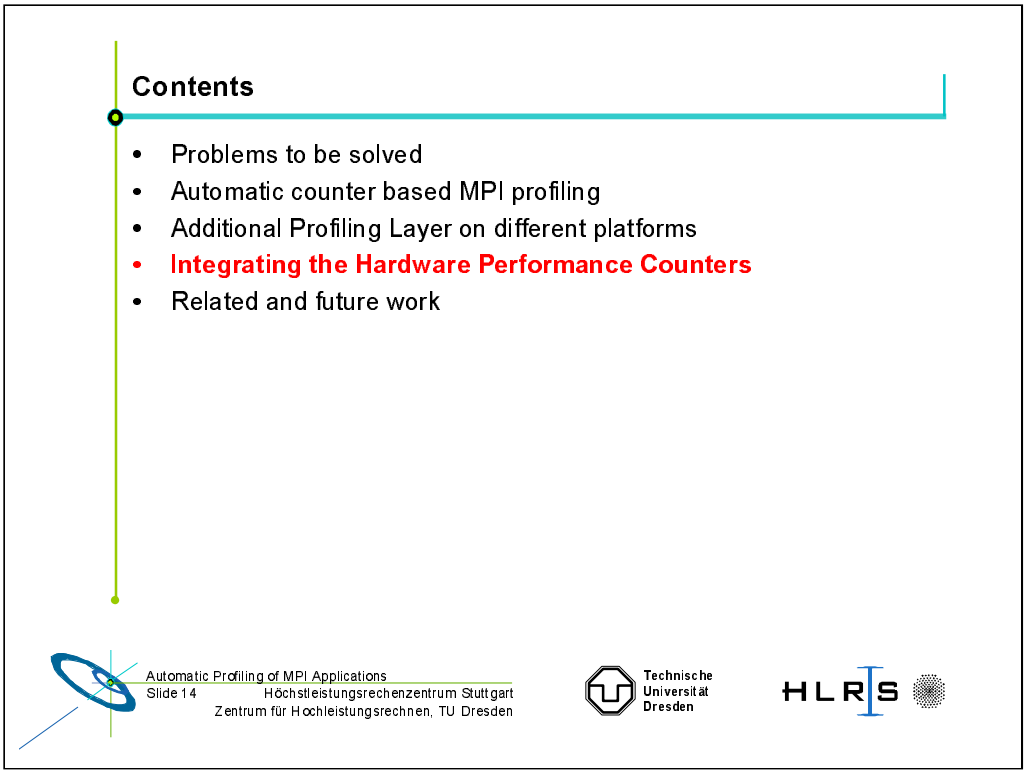
It also writes a statistic about the whole system.



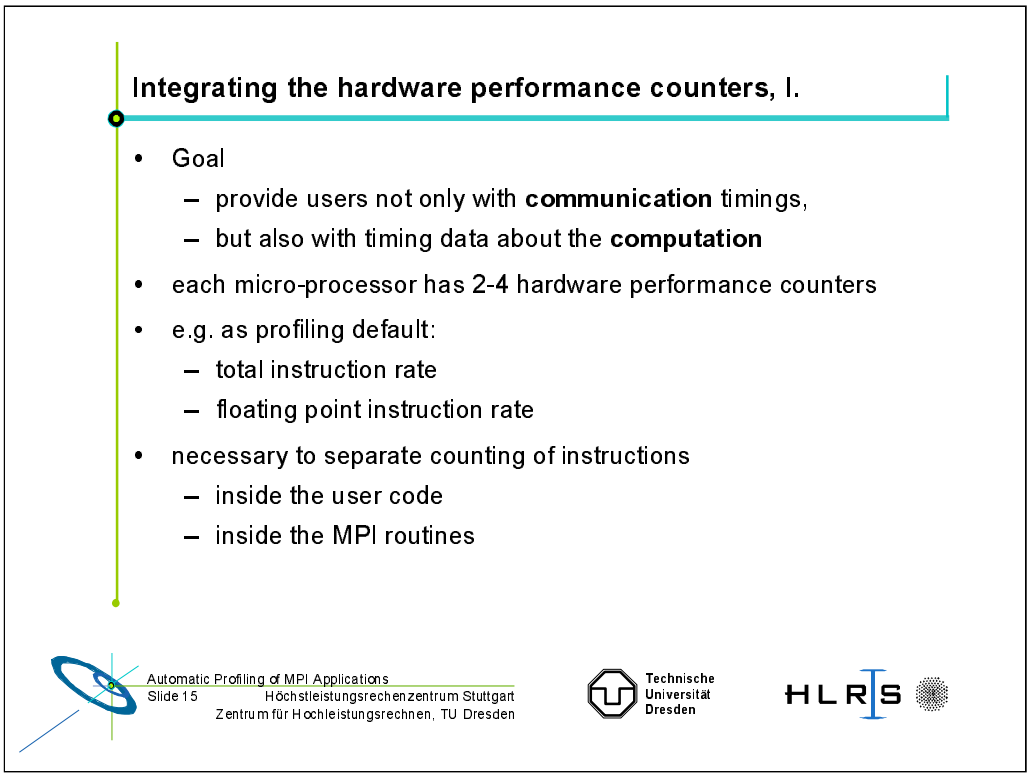
• The full design of this instrumentation is too complex to be presented in this short talk. It is described in detail in the proceedings.

• The requirements are
  - a fast subroutine-call mechanism,
  - and a fast clock-routine,
  - and a fast access to the hardware performance counters.

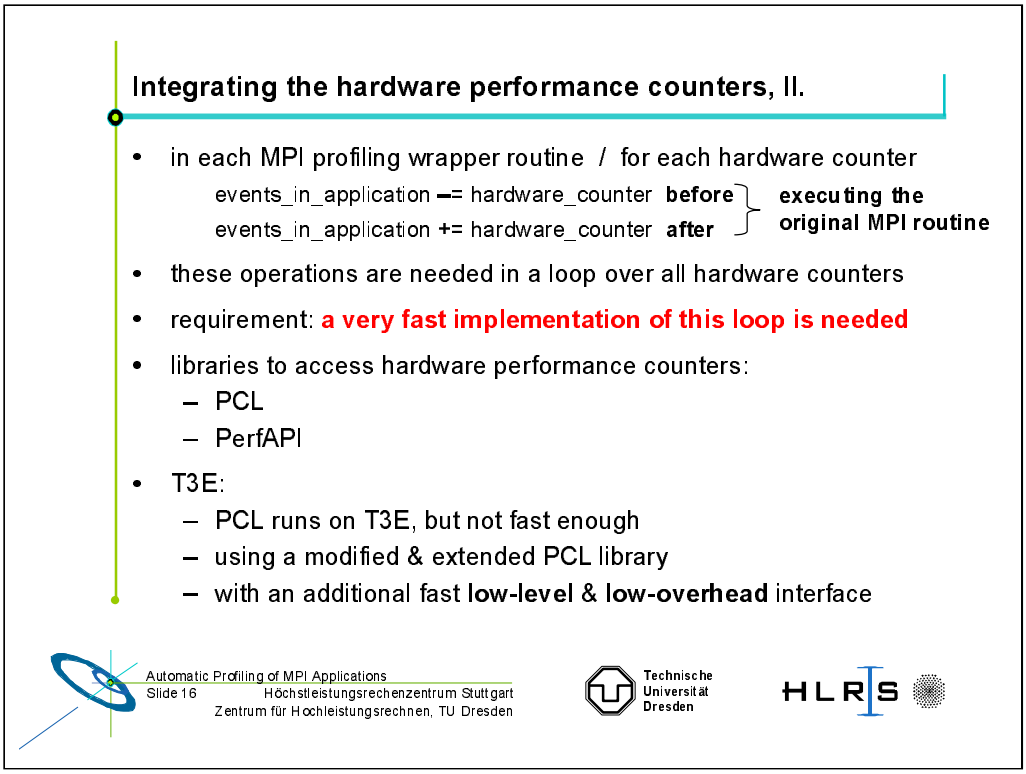
• The automatic counter profiling is portable to each system that meets these requirements.



Our next topic is the integration of the hardware performance counters.

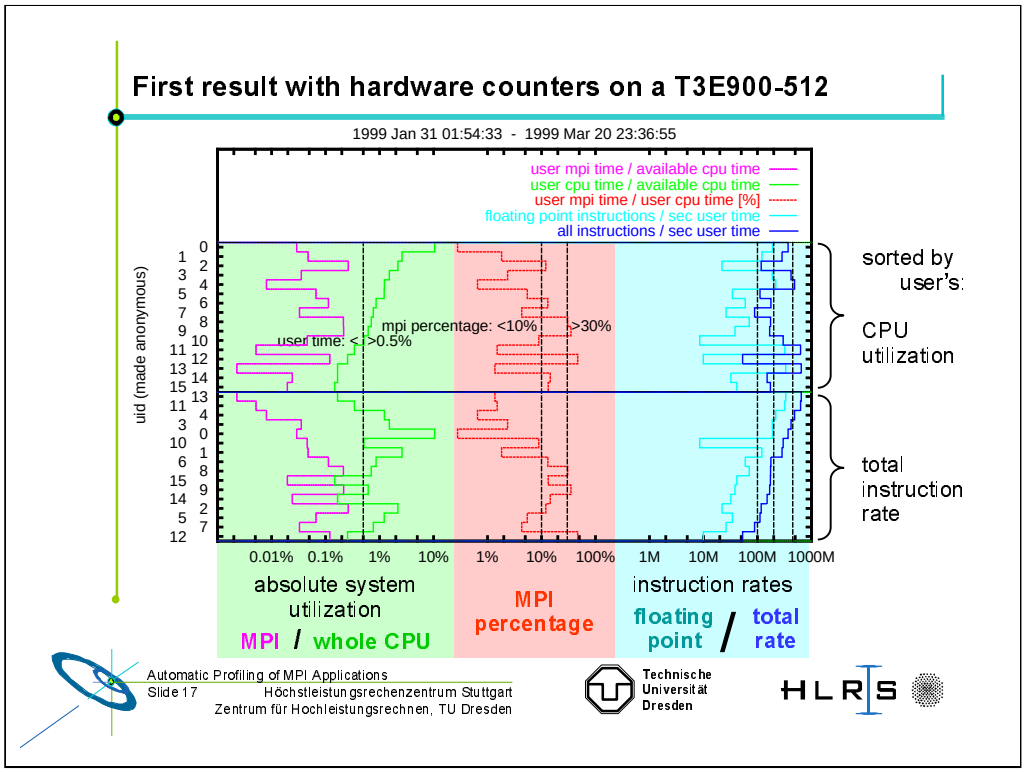


- The major goal of integrating the hardware performance counters is
  - to provide the users not only with the communication timings,
  - but also with timing data about the computation.
- Each micro-processor has 2, 3 or 4 hardware performance counters. For example on a T3E, the first counter can be used to count any executed instruction, the second counter can be used to count all floating point instructions and the third for counting level 1 or 2 data cache misses.
  From a longer list of possible events, one can choose 3 events that should be counted by the 3 hardware performance counters on the T3E.
- For the MPI profiling, the default is
  - the total instruction rate
  - and the floating point instruction rate
- Because MPI internally implements any wait as a polling busy wait, it is necessary to separate the counting of the instructions
  - executed inside the user-code
  - from the huge amount of instructions issued inside of MPI routines while doing nothing else than polling.

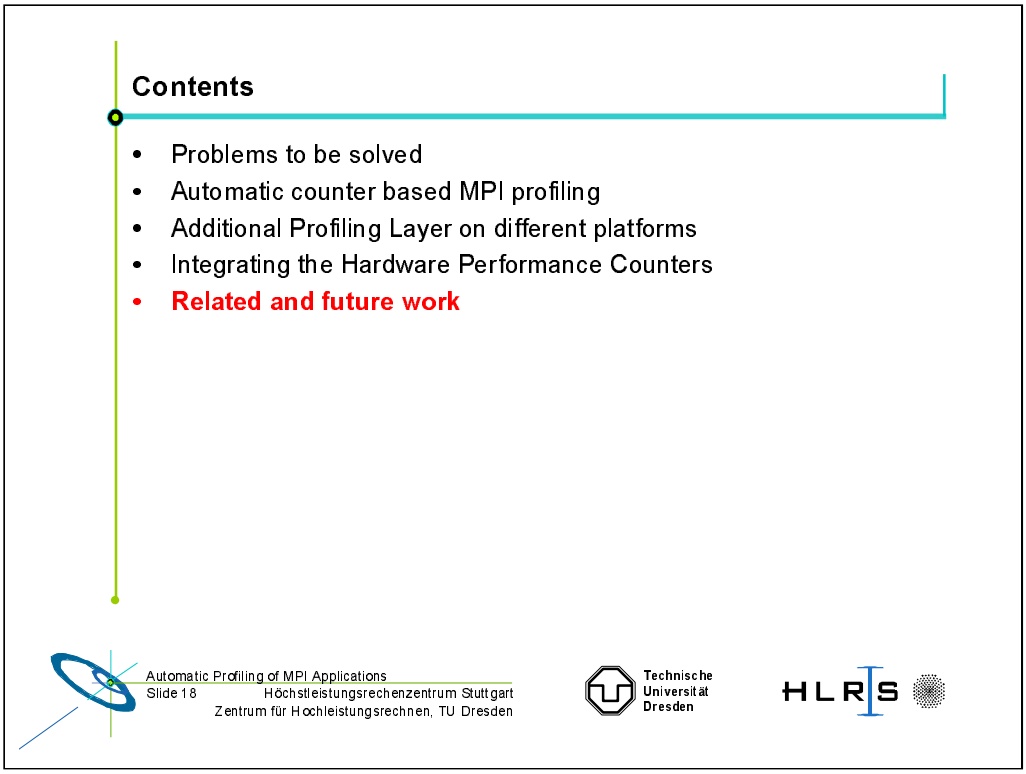


To separate the hardware counting between user-code and MPI routines we have to execute

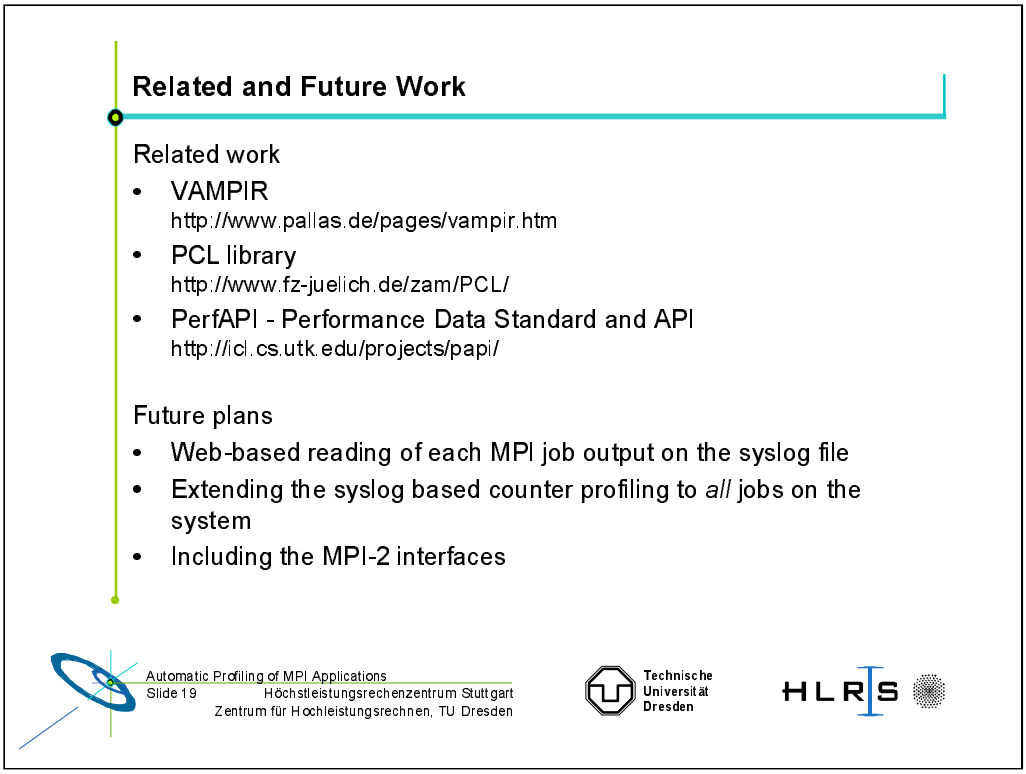
- in each MPI profiling wrapper routine
  - a subtraction of the current status of the hardware counters before the original MPI routine is executed,
  - and a sum after the original MPI routine has finished.
- These operations are needed in a loop over all hardware counters.
- Because the whole work of the profiling wrapper should not consume more than the tenth of the MPI latency, a very fast implementation of this loop is needed.
- There exist at least two libraries to access hardware performance counters:
  - the PCL library from the Research Center Jülich,
  - and PerfAPI, a ptools standard and library
- Now on our T3E:
  - PCL runs on T3E, but it was not fast enough.
  - Therefore I am using a modified & extended PCL library
  - with an additional fast low-level and low-overhead interface.



- This slide shows a first 49-day-statistics with the hardware performance counters.
- Each line of the diagram represents one user.
- The upper and the lower part show the same users, but the upper part is sorted by their CPU utilization and the lower part by their total instruction rate.
- Looking first at the upper part, we see in the green area the system utilization in percent of the whole available CPU time.
- The green line shows the CPU utilization of each user.
  The vertical dashed bar marks 0.5 percent of the total system. At least the upper 11 users have production jobs running on our system.
  The purple line shows the time spent in MPI routines.
- The red line in the red area shows the ratio of MPI time to CPU time. The vertical bars mark a MPI percentage of 10 and 30 percent.One can see that half of the users are in or nearby of this interval and the other users have only a minimal MPI overhead.
  The green and red line show that the production users have a moderate MPI percentage.
- For looking at the instruction rates we switch to the lower part of the picture.
  The turquoise line shows the floating point instruction rate of the application part. On a T3E, each instruction can issue one or two operations. Therefore only the floating point instruction and not the operation rates can be measured.
- The vertical bars mark 100, 200 and 450 Mega floating point instructions per second. 450 is the theoretical peak.One can see that the floating point instruction rates differ between 9 and 333 Mega floating point instructions per second.
- The blue line shows the total instruction rate of the application part. In both lines the MPI instructions are excluded. One can see that most users have more than 100 million instructions per second, also those user that have an MPI percentage of 30 percent.



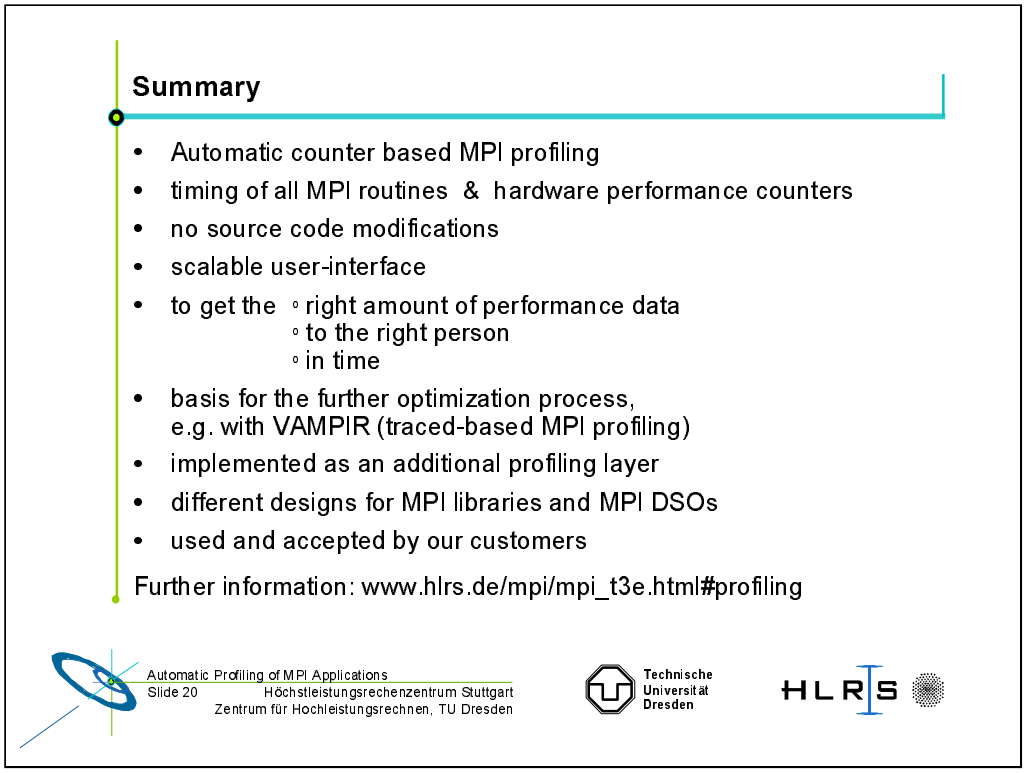
Now the last two slides.



As related work, I want to mention VAMPIR, PCL and PerfAPI.

Our future plans are:

- a web-based reading of the profiling data on the syslog file,
- extending the syslog based counter profiling to all jobs on the system,
- and including the MPI-2 interfaces.



Now we come to the summary.

- I gave an overview on our automatic counter based MPI profiling.
- It records the time spent in all MPI routines and it records the hardware performance counters.
- No source code modifications are necessary for the users.
- A scalable user-interface helps
- to get the
  - right amount of performance data
  - to the right person
  - in time
- The automatic counter profiling is a basis for the further optimization process, for example with traced-based profiling tools.
- It is implemented as an additional profiling layer.
- For this, I have shown two different designs for Unix libraries and for dynamic shared objects.
- The automatic counter profiling is used and accepted by our customers in Stuttgart and Dresden.
- Further information is available under this URL.

Thank you for your interest.