Published in the proceedings of the EuroPVM/MPI'99, Barcelona, Spain, Sept. 26-29, 1999.

Automatic Profiling of MPI Applications with Hardware Performance Counters

Rolf Rabenseifner[1]

Center for High Performance Computing (ZHR), Dresden University of Technology
Zellescher Weg 12, Willers-Bau A 117, D-01062 Dresden, Germany
www.tu-dresden.de/zhr/, rabenseifner@rus.uni-stuttgart.de

Abstract. This paper presents an automatic counter instrumentation and profiling module added to the MPI library on Cray T3E and SGI Origin2000 systems. A detailed summary of the hardware performance counters and the MPI calls of any MPI production program is gathered during execution and written in MPI_Finalize on a special syslog file. The user can get the same information in a different file. Statistical summaries are computed weekly and monthly. The paper describes experiences with this library on the Cray T3E systems at HLRS Stuttgart and TU Dresden. It focuses on the problems integrating the hardware performance counters into MPI counter profiling and presents first results with these counters. Also, a second software design is described that allows the integration of the profiling layer into a dynamic shared object MPI library without consuming the user's PMPI profiling interface.

Keywords. MPI, Counter Profiling, Instrumentation, Hardware Performance Counters, Trace-based Profiling, PerfAPI, PCL.

1 Counter-Based Profiling

Today, job accounting on MPP hardware platforms does not provide enough information about the computational efficiency or about the efficiency of message passing (MPI) usage either to the users or to the computing centers. There is no information available about bandwidth and latency or integer and floating point operation rates achieved in real application runs. Therefore, users and hotline centers have no reliable information base for technical and political decisions with respect to programming and optimization investment. Existing trace-based profiling tools are too complicated for a first glance at an application and can be used in small test-jobs only, not in long-running production jobs.

To solve this problem, the High-Performance Computing-Center (HLRS) at the University of Stuttgart has combined the method of counter-based profiling with the technics of writing system log-files. For each MPI routine, the number

[1] The author is an employee of the High-Performance Computing-Center (HLRS) at the University of Stuttgart (www.hlrs.de/people/rabenseifner/). Most of this work was done while the author was a visiting research associate at the ZHR from January to April 1999.

of calls, the time spent in the routine and the number of transferred bytes are written at the end of each parallel job to a syslog file of the computing center and, optionally, to a user file. The integration of the PCL library [1] allows the automatic instrumentation with the microprocessor's hardware performance counters (e.g. floating point instructions) to get information about the computational efficiency of each program. With that, the user has a criterion whether tuning the numerical part or the communication part promises greater benefit.

An analysis tool reads the syslog file and, on a weekly basis, sends a summary to each user about her/his jobs and writes a web-based summary for the computing center. The results of the first half-year on CRAY T3E 900-512 at the HLRS are presented in [8]. In a survey, our users showed that in the past, the profiling information was used only seldom for tuning the individual applications because the profiling tool was only available after the application development was finished and production was started. But 75 % of those interviewed believe that the profiling can help in the future to improve their applications [9].

The profiling was implemented, tested and installed as default library on the T3E systems at HLRS Stuttgart and TU Dresden, and it is now ported to the Origin2000 at TU Dresden. The counter-based profiling only has a minimal overhead. The memory requirements on a T3E-900 are 200 kBytes. The counting requires 0.3 - 0.5 μsec per MPI call and writing the syslog file requires about 0.1 sec for each job. The overhead was 0.03 % of the application CPU time on the first half-year average in Stuttgart. Including the hardware counters, the overhead is about 300 kBytes memory, 2 μsec/call and about 0.1 - 0.2 % (expected) in all.

The PCL library was developed by the Forschungszentrum Jülich. For integrating the PCL library, the hardware counters' reading routine of the PCL library on the T3E has been optimized from about 35 μs to about $0.5 - 1.0$ μs by removing the operating system calls. This allows differentiation between counting hardware events inside and outside of the MPI routines. This is important because otherwise some hardware counters (load, integer instruction, any instructions) could not be used for measuring the user application since the busy wait operations of MPI would inflate their values.

2 Software Design

To use the instrumented MPI library as default library, it is necessary to export the full MPI and PMPI interface for Fortran and C, as described in the MPI standard and implemented in the public and vendor's MPI libraries. This means that it was not possible to consume the PMPI profiling interface for the intended instrumentation, i.e. a method had to be used that allowed two profiling layers. The Figures 1 and 2 show the different software designs for Unix libraries (archive libmpi.a) and dynamic shared objects (DSO libmpi.so).

For Unix libraries, the instrumentation is implemented as one wrapper routine to each MPI routine and added to the original MPI library, in which the original routines' names are modified with a binary-file editor. This interface was developed for a CRAY T3E system and is described in [8].

For DSOs, the new MPI-DSO libmpi.so contains only the instrumented wrappers and an initialization routine that binds the (up to this time) unresolved

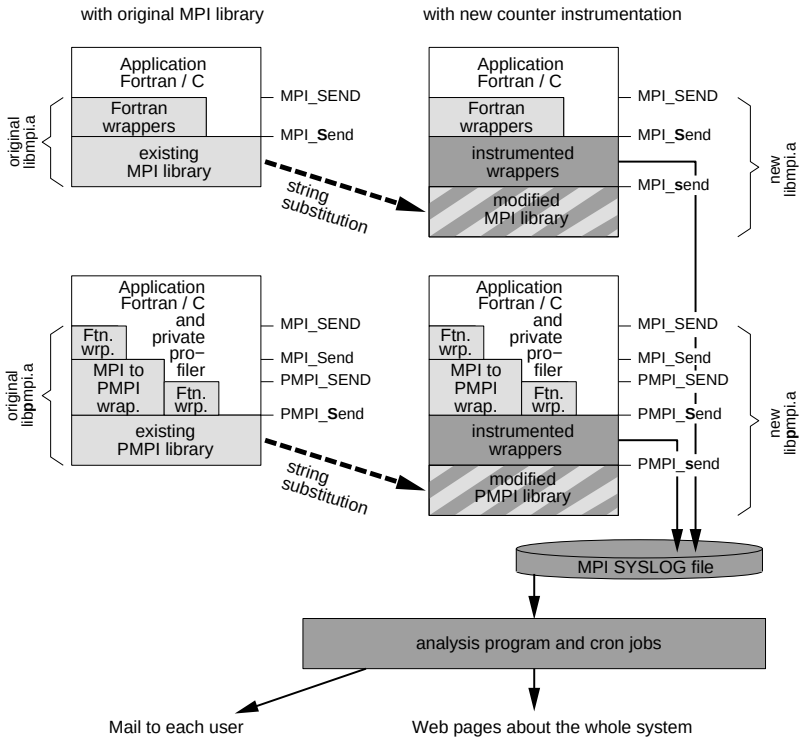

Fig. 1. The software design for MPI libraries (libmpi.a)

MPI references of the wrapper routines with dlopen and dlsym to the original MPI-DSO at start-up time. For this, the original MPI-DSO library was renamed libMpi.so. The dynamic linking at start-up time is necessary because static linking is impossible since the entry-point names of the instrumented wrapper routines and of the original MPI library routines are identical. This design requires that the original MPI routines never internally call other MPI routines. This is the case for the SGI MPI library. For applications written in C, this design adds only one additional subroutine call and the instrumentation.

For applications written in Fortran, the design depends on the implementation method of the Fortran MPI language binding. If a Fortran MPI routine is implemented as a Fortran-to-C wrapper routine that calls its C counterpart, then an empty wrapper for this Fortran interface must be added to the new libmpi.so and the dynamic linking establishes the following calling stack: application → new empty Fortran wrapper → original Fortran-to-C wrapper → new instrumented C wrapper → original MPI C interface. Compared with C, this case costs one additional subroutine call. If a Fortran MPI routine is implemented directly, then the Fortran wrapper in the new libmpi.so must be instrumented like the C wrapper, and there are no additional costs. This case may be necessary for routines with arguments that are externals or for optimized MPI routines. Additional wrappers must be included for the three special argu-

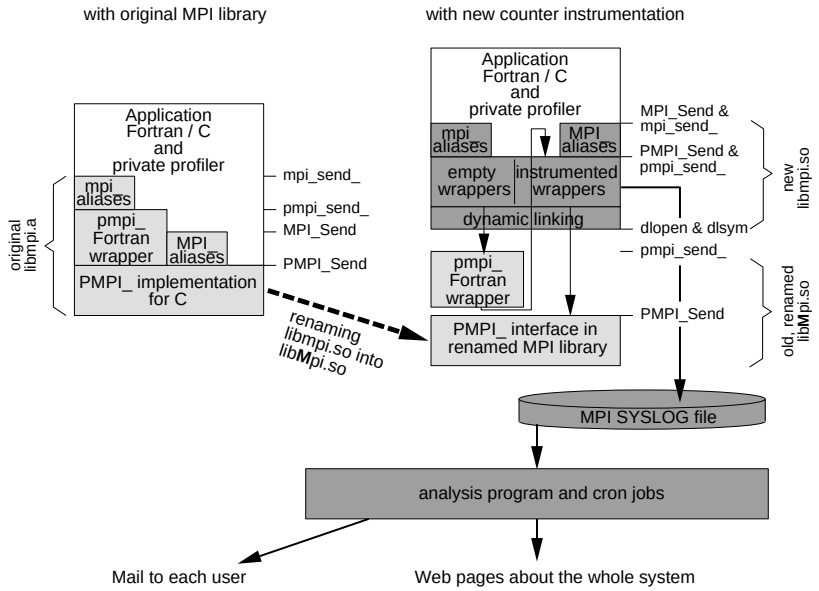

Fig. 2. The software design for MPI dynamic shared objects (libmpi.so)

ment values MPI_NULL_COPY_FN, MPI_DUP_FN and MPI_NULLL_DELETE_FN in Fortran and C. This design was developed for SGI IRIX 6.5. The library is portable to any system with fast subroutine calls and a fast local clock routine.

3 Adding Hardware Performance Counters

Each micro-processor has implemented some (2-4) hardware performance counters that are able to count one type of event among a given larger set of hardware events, e.g. completed instructions, floating point instructions, loads, stores or cache misses. The PCL library [1] developed by the Forschungszentrum Jülich is a common interface for currently 6 different micro-processors and access methods of the operating systems. PerfAPI [5] is a standardization effort of The Parallel Tools Consortium [11] to achieve a common interface to access the hardware performance counters.

To use these counters to measure the applications' efficiency, it is necessary to separate the hardware events generated by the application code and by the MPI library routines. This is not necessary for events that are generated very seldom by the MPI routines, such as *floating point instructions*, but the separation is absolutely necessary for events that are often generated by MPI, such as those *load instructions* used in the busy-wait implementation of MPI_Receive. To separate MPI and application events, it is necessary to have an extremely fast access to the hardware counters and to minimize data cache misses inside the instrumented wrappers (e.g. by minimizing any access to global variables).

Typically the operating system exports one of two different interface types: a) the hardware counters can be *read* like a clock, i.e. they are not reset after

reading, and b) they can be *read out*, i.e. they are always reset to zero after reading them. For both interfaces, the instrumented wrappers have to implement one integer operation for each hardware counter before calling the original MPI routine and another one after returning from it:
In case a) events_in_mpi $-=$ counter must be issued before each MPI call and events_in_mpi $+=$ counter after its return and events_in_application $=-$counter at the beginning and events_in_application $+=$ (counter $-$ events_in_mpi) at the end of the whole application.
In case b) events_in_application $+=$ counter must be issued before each MPI call and events_in_mpi $+=$ counter after its return and counter $=$ events_in_mpi $=$ events_in_application $= 0$ at the beginning and events_in_application $+=$ counter at the end of the whole application. This means that the major requirement of a common **low-level** interface to access the micro-processors' hardware counters is that the two operations plus and minus must exist in the form

$$\begin{array}{c}\text{for (i=0; i<number\_of\_hardware\_counters; i++)}\\ \text{local\_event\_counter[i]} \pm= \text{hardware\_counter[i]}\end{array} \qquad (1)$$

and also the information must be available whether this is a *read* or *read out*.

Unfortunately, neither the PCL library nor the PerfAPI interface design meet this requirement. E.g., the PCL library only has a *read out* interface, which adds additional operations, cache misses and resets of the hardware counter if the hardware supports the *read* interface. Additionally, PCL only has a high level interface that implements a matrix operation on the set of hardware counters and that implies at least an additional load/store for each counter. This matrix operation implements the mapping of the counters defined by PCL to any hardware counter or any difference or sum of several hardware counters. The matrix operation would be done twice for each MPI wrapper call, if PCL were used, instead of only once at the end of the application, which happens if the method (1) is used and the matrix operation is done only once before writing the results of events_in_mpi and events_in_application to the syslog-file. To integrate the hardware performance counters into the automatic MPI profiling, we have added an interface to the PCL library for the CRAY T3E that is similar to the required interface (1). With this and by removing all unnecessary operating system calls, the time to access the hardware performace counters could be reduced from 35 μs to about 0.5-1.0 μs.

4 First Results with Hardware Counters

Fig. 3 shows the users' profiles in the first seven weeks we used the hardware performance counters. Each row represents one user. The upper and lower part plot the same information, but differently sorted. The upper part is sorted by the CPU time consumed by each user, and the lower part by the total instruction rate. Each part combines three different plots:

The solid line in the left diagram represents the CPU time each user has consumed as part of the total time of the whole system in the analyzed time interval. The users are sorted by this value. The vertical bar marks 0.5 % of the

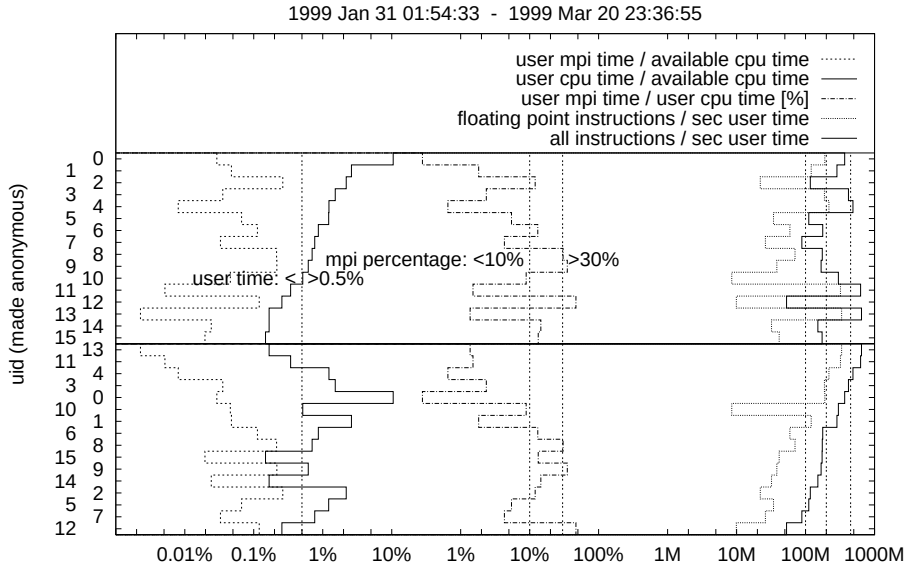

Fig. 3. Users' profiles, sorted by CPU time (upper part) and by instruction rate (lower part) – description see Section 4

total system. The figure represents the top 16 users of the HLRS that have used the new MPI library which was now instrumented with the hardware counters. In total, the 16 users have consumed 24.3 % of the whole system. The left dashed line shows the percentage of the time the user applications have consumed in MPI routines.

The diagram in the middle shows the ratio of MPI time to application time. The vertical bars mark a ratio of 10 and 30 %. The MPI percentage varies between 0.3 % and 46.8 %. On average, the MPI percentage was 5.2 %.

The diagram on the right shows the floating point instruction rate (dotted line) and the total instruction rate (solid line). The numbers are averages referring to one processor. On a T3E, each floating point instruction can execute one or two floating point operations. The floating point operation rate cannot be measured, but lies between the floating point instruction rate and twice the floating point rate. About half of the users' application runs could be used to analyse the hardware counters. The other jobs could not read the counters because the application was not yet relinked with the new library or else the partition was moved to other processors during execution. The floating point instruction rate of these 16 users varies between 9 and 333 MFL_instructions/sec (MFLips); weighted with the CPU time, the average is 138 MFLips, which implies that the MFLOP rate is between 138 and 276 MFLOPS, i.e. between 15 and 30 % of the peak performance of 900 MFLOPS. The vertical bars mark 100, 200 and 450 MFLips. The total instruction rate of the application code, except the MPI routines (solid line), is computed by dividing the number of instructions in the whole application minus the number of instructions executed in the MPI

routines by the whole execution time. The total instruction rate varies between 53 and 647 M_instructions/sec.

The upper part of the picture helps to review the efficiency of the most relevant users. The lower part of the picture gives an insight into the correlation of MPI percentage and instruction rate. The numbers presented for each user (i.e. the MPI percentage, the floating point instruction rate and the total instruction rate) are a major information base for decisions with respect to programming and optimization investment since these numbers give a good overview of the achieved efficiency in computation and communication. The analysis tool sends these numbers and additional details to each user in a weekly mail.

5 Related and Future Work

[1] describes the hardware counter library PCL. [5] is a standardization effort for accessing of the hardware performance counters. [10] is a comprehensive overview about monitoring and profiling systems. Trace-based profiling and analysis is described in [2, 3, 6, 7]. [4] describes a local, user-callable MPI counter profiling. [8] focuses on the global view of the counter-based MPI profiling and includes the statistical results of half a year of profiling nearly all MPI applications running on a CRAY T3E 900-512, but without instrumenting the applications with the hardware counters. [9] decribes the scalability of the profiling user interface.

We hope that the results of this project can influence the standardization effort of PerfAPI and that the new low-level interface for PCL can be implemented on all supported hardware platforms in an efficient way. Also, it is planned to use the hardware counter profiling for *all* applications and not only for MPI applications, although there is no plan to differentiate then between hardware events issued by the application code and those issued by non-MPI communication (e.g. with PVM, HPF or shmem). We will generate global half-year statistics of the CRAY T3E 900-512 used by the HLRS Stuttgart, similar to that in [8] but including the hardware counters. The major goal is to extend this profiling to all terminating correctly applications to see the hardware performance counters not only on MPI applications. The computing center has planned to use the profiling results to offer the users help in optimizing their individual applications. In a survey, we saw that 85 % of our users would like to be addressed for that reason [9]. Extending the profiling interface for OpenMP applications is under investigation.

6 Conclusion

This project shows that combining the methods of counter profiling, job accounting and accessing the hardware performance counters can give more insight into the users' applications than achievable by previously used tools with similar costs. The paper has shown two different methods to integrate an additional profiling level into existing MPI archives and dynamic shared objects without losing the standardized MPI profiling interface for other profiling tools. The automatic MPI profiling is a method to get enough information to decide whether

the application is running as expected, one more chance to detect major bottlenecks and a basis to decide whether trace based tools should be used to optimize the communication pattern or whether direct hardware counter instrumentation should be applied to optimize the computational part.

Acknowledgments

The author would like to acknowledge his colleagues at ZHR and HLRS and all the people that supported this project with suggestions and helpful discussions. He would like to thank especially R. Berrendorf for his support of the PCL library, W.E. Nagel for productive discussions and for the invitation to Dresden, R. Rühle for giving major resources for this project, and M. Heine and E. Salo for the hints on SGI's MPI/DSO.

References

1. R. Berrendorf and H. Ziegler, *PCL - The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors*, internal report FZJ-ZAM-IB-9816, Forschungszentrum Jülich, Oct. 1998. http://www.fz-juelich.de/zam/docs/autoren98/berrendorf3.html
2. M. T. Heath, *Recent Developments and Case Studies in Performance Visualization using ParaGraph*, Proceedings of the Workshop Performance Measurement and Visualization of Parallel Systems, G. Haring and G. Kotsis (ed.), Moravany, Czechoslovakia, Oct. 1992, p. 175-200.
3. V. Herrarte and E. Lusk, *Studying Parallel Program Behavior with Upshot*, Argonne National Laboratory, technical report ANL-91/15, Aug. 1991
4. *HP* MPI *User's Guide*, 4.1 Using counter instrumentation, HP, B6011-90001, Third Ed., June 1998.
5. P. J. Mucci, S. Browne, G. Ho and C. Deane, *PerfAPI - Performance Data Standard and API*. http://icl.cs.utk.edu/projects/papi/
6. W. E. Nagel et al., *VAMPIR: Visualization and Analysis of MPI Resources*, Supercomputer 63, Volume XII, Number 1, Jan. 1996, pp. 69-80. http://www.kfa-juelich.de/zam/docs/autoren95/nagel2.html and Technical Report KFA-ZAM-IB-9528, Jan. 1996 ftp://ftp.zam.kfa-juelich.de/pub/zamdoc/ib/ib-95/ib-9528.ps
7. W. E. Nagel and A. Arnold, *Performance Visualization of Parallel Programs: The PARvis Environment*, technical report, Forschungszentrum Jülich, 1995. http://www.fz-juelich.de/zam/PT/ReDec/SoftTools/PARtools/PARvis.html
8. R. Rabenseifner, *Automatic MPI Counter Profiling of All Users: First Results on a CRAY T3E 900-512*, Proceedings of the Message Passing Interface Developer's and User's Conference 1999 (MPIDC'99), Atlanta, USA, March 1999. www.hlrs.de/people/rabenseifner/publ/publications.html
9. R. Rabenseifner, S. Seidl and W. E. Nagel, *Effective Performance Problem Detection of MPI Programs on MPP Systems: From the Global View to the Detail*, Parallel Computing '99 (ParCo99), Delft, the Netherlands, August 1999. www.hlrs.de/people/rabenseifner/publ/publications.html
10. M. van Riek, B. Tourancheau and X.-F. Vigouroux, *Monitoring of Distributed Memory Multicomputer Programs*, University of Tennessee, technical report CS-93-204, and Center for Research on Parallel Computation, Rice University, Houston Texas, technical report CRPC-TR93441, 1993. http://www.netlib.org/tennessee/ut-cs-93-204.ps and ftp://softlib.rice.edu/pub/CRPC-TRs/reports/CRPC-TR93441.ps.gz
11. The Parallel Tools Consortium – www.ptools.org