EFFECTIVE PERFORMANCE PROBLEM DETECTION OF MPI PROGRAMS ON MPP SYSTEMS: FROM THE GLOBAL VIEW TO THE DETAILS

ROLF RABENSEIFNER

*High-Performance Computing-Center Stuttgart (HLRS),**
University of Stuttgart, D-10550 Stuttgart, Germany
www.hlrs.de/people/rabenseifner, rabenseifner@hlrs.de

PETER GOTTSCHLING, WOLFGANG E. NAGEL AND STEPHAN SEIDL

Center for High Performance Computing (ZHR),
Dresden University of Technology, D-01062 Dresden, Germany
www.tu-dresden.de/zhr, [gottschling|nagel|seidl]@zhr.tu-dresden.de

This paper presents an automatic counter instrumentation and profiling module added to the MPI library on Cray T3E systems. A detailed summary of the hardware performance counters and the MPI calls of any MPI production program is gathered during execution and written on a special `syslog` file. The user can get the same information on 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 scalability aspects of the new interface: How to obtain the right amount of performance data to the right person in time, and how to draw conclusions for the further optimization process, e.g. with the trace-based profiling tool Vampir.

1 Counter-Based Profiling

Today, job accounting on MPP hardware platforms does not provide enough information about the computational efficiency nor about the efficiency of message passing (MPI) usage for both, users and computing centers. There is no information available on bandwidth and latency or integer and floating point operation rates obtained 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. For a first glance at an application, the existing trace-based profiling tools are too complicated. They can be used in small test jobs only, but 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 pro-

*Part of the work was done while the author was a visiting research associate at ZHR, Jan.-Apr. 1999.

filing with the techniques of writing system log-files. For each MPI routine, the number 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 at 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 receive information about the computational efficiency of each program.[10]

An analysis tool reads the syslog file and, on a weekly basis, sends a summary to each user about his jobs and writes a web-based summary for the computing center. The results of the first half-year have shown the MPI usage on the CRAY T3E 900-512 at the HLRS.[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 in the first half-year average in Stuttgart. Including the hardware counters, the overhead is about 300 kBytes memory, 2 μsec per call and about 0.1 - 0.2 % (expected) in all.

2 Scalable User-Interface

The user interface is designed so that the cost-benefit ratio is scalable for the user. This means that one is able to receive the most important information about an MPI application with an effort that is nearly zero, and if more information is wanted, there are several levels of support:

Level 1: Reading the first lines of an e-mail yields the MPI percentage, the instruction rate of all completed hardware instructions, the floating point instruction rate, and the level 2 cache miss rate of the jobs in the last week, if they have used the MPI default library.

Level 2: Reading the details of the e-mail yields for each MPI routine a summary of the calling count, execution time, and transferred bytes. Additionally, an estimate of the transfer and synchronization parts of the execution time is given. For each hardware performance counter, the total number of events and the rates are printed for the application part and for all MPI routines together.

Levels 1 and 2 could be implemented because the instrumentation is added to the default MPI library and writes the counters via the syslogd daemon to a syslog file.

Level 3: The user has to set a special environment variable in the MPI job before calling mpirun. As a consequence, the user gets all the information mentioned above (except the synchronization part estimate) for each parallel application run that uses the environment variable. For the future it is planned to implement a Web interface for the database created upon the syslog information. Then the user can view each job without setting the environment variable, and for levels 1-3 the user does not need to modify anything in the programs or job commands.

Level 4: The application can call MPI_Pcontrol to print out the state of the counters at any location inside the application. This gives all the information not only at the end of the application run, but also at any time and on each individual process (and not only the average of all processes).

Level 5: Setting one more environment variable, the user can choose other hardware performance counters, e.g. the integer instruction rate, level 1 data cache hits, and misses.

The levels 1 to 5 provide the information base to decide whether other tools should be used. Therefore, the next level is not part of the automatic MPI profiling, but part of the scalability strategy:

Level 6: Using trace based tools, e.g. Vampir, the user gets detailed insight into the computation and communication structure of the application. Using the PCL library directly or using other performance visualization tools can give a detailed insight into the computation efficiency and any problems of the application. PAT and Apprentice can also be used to locate and analyze the computational kernel of the application.

One of the key issues for this design is that on average the instrumentation overhead is in the range of a thousandth of the CPU time consumed by the applications. 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 or direct hardware counter instrumentation should be applied.

3 Comparing Counter-Based and Trace-Based Methods

The major differences between the counter and trace-based profiling can be seen by comparing the following characteristics: *Automatic counter-based MPI profiling* analyzes the whole MPI application. The analysis of long-running production jobs is not a problem. The extent of instrumentation must be restricted in time and memory to about a thousandth of the application because the tool is used for all MPI application runs. Only a small amount of data is written for each application run on the system's disk. The computing center and the users can get an overview about all jobs parallelized with MPI.

Trace-based profiling can normally be used to analyze short test jobs only. The instrumentation overhead can vary within the range of 5 to 10 percent because the user decides whether or not the tool is applied. Each trace can produce a large amount of data, written on the user's disk. The computing center does not get any information and long running parallel jobs can not be viewed, although these jobs are more important because they consume most of the computing resources.

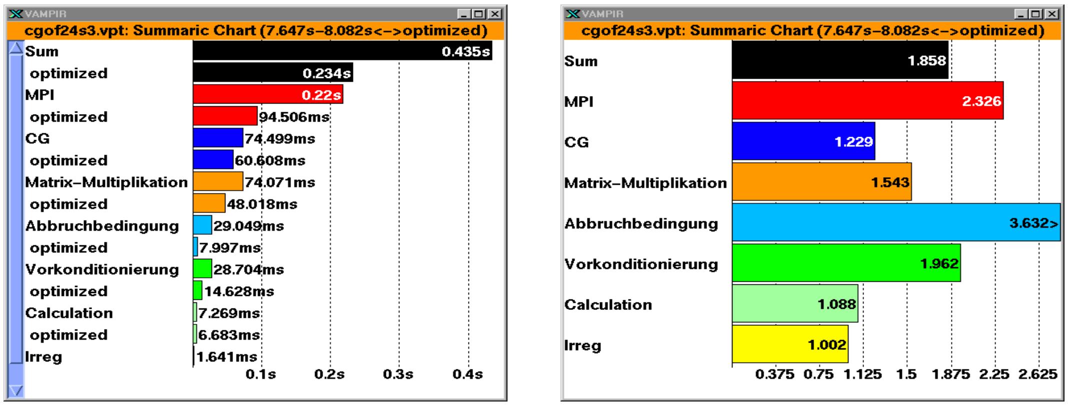
4 Optimizing User Applications

In the case that the reported application performance does not correspond to the user's expectation, the per-job counter-based profiling output should also be examined. From the user's point of view, typical results of counter-based profiling are: 1) The application does or does not spend a lot of time in MPI routines. 2) The application might not seem to be load-balanced since a lot of time is spent inside of MPI receives, barriers or collective operations. 3) The overall and the user part MFLOP rates are or are not as expected.

To localize such effects of MPI routines, or to figure out other critical parts, the application can be instrumented to be analyzed by means of Vampir, for example. For instrumenting codes, there are different options, depending on the particular platform. On Fujitsu, Hitachi, and NEC systems, it is sufficient to switch on an appropriate compiler flag. On Cray T3E, the software tool PAT can do this job. In general, the Vampir visualization and trace system from Pallas GmbH will help or, in order to solve special tracing problems, one could contact the TU Dresden for an alternative binary trace library.

An example: At our center, the relatively low floating-point rate associated with a user who is investigating linear solvers based on a new algebraic cascadic conjugate gradient method (ACCG) has induced more detailed studies. Besides the floating-point rate, the level-1-analysis showed that the ratio of the total weekly MPI-time to the corresponding application time was 0.328. With respect to a multigrid-like application where a significant part of the computing time is spent on small linear equations, this result seems to be acceptable.

The examination on level 2 displayed that most of the 9640 MPI-seconds were consumed by three routines: MPI_Bcast with 5120 s, MPI_Alltoallv with 1410 s and MPI_Allreduce with 1270 s. For all three functions, the number of calls and transferred bytes did not explain this time consumption. Instead, it



(a) Comparison of the two versions of the solver (without initialization)

(b) Quotient: Original/optimized time

Figure 1. Parts of the ACCG solver

might be an indicator for a load-imbalance.

On level 3, we considered the per-job statistic of a test case which solves a linear equation with 253,000 unknowns on 24 processors. The test case computes the stationary solution of a ground-water flow simulation instead of the in-stationary solution. In the investigated job, the same three MPI-functions took a significant part of the computing time: MPI_Bcast 6.323 s, MPI_Allreduce 163 ms and MPI_Alltoallv 142 ms. The trace-based analysis showed that the time spent on MPI_Bcast originates in reading the input files on one processor. In this phase of the program, all other processors are waiting at MPI_Bcast. This fact explains the large idle times of this function in the weekly statistic.

Since the discretization of the differential equations and therefore, the construction of linear equations is not part of our research area, we are solely interested in the run-time behavior of the linear solver (i.e. the ACCG solver). With MPI_Pcontrol (level 4), we got the same counters as in the per-job statistic, but now at the beginning and the end of the linear solver. The differences of the corresponding counters in the two snapshots indicated that only 0.435 s of the 8 seconds are spent in the solver. Moreover, we saw that MPI_Bcast only took 0.2 ms during the computation of the solver. Unfortunately, most of the time of MPI_Allreduce (159 ms) falls in this phase. Since MPI_Alltoallv required only 60 ms in the solver it seemed most promising to optimize the allreduce computation.

Replacing the call of MPI_Allreduce with a function using a similar shmem-routine, the allreduce computing time could be reduced from 159 ms to 36 ms. For a better comparison, we grouped the new function to MPI in

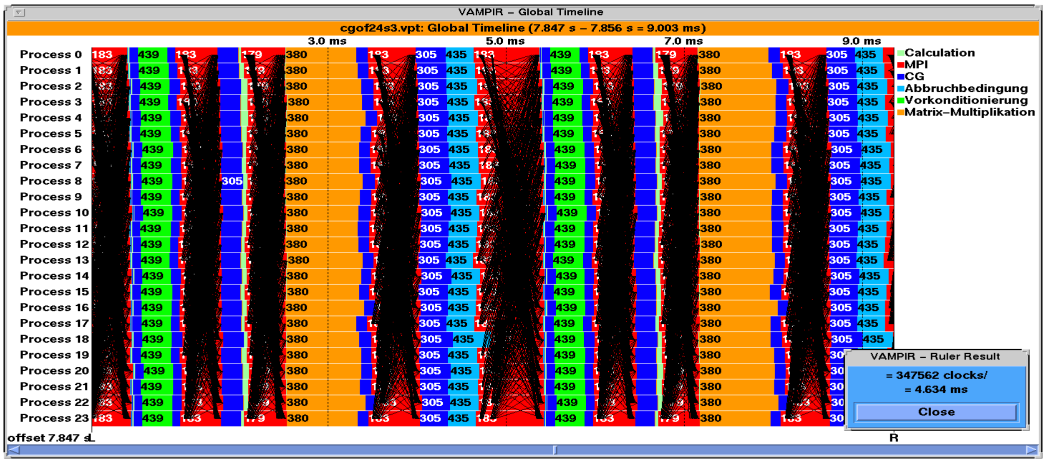


Figure 2. Two iterations of the conjugate-gradient solver (original implementation)

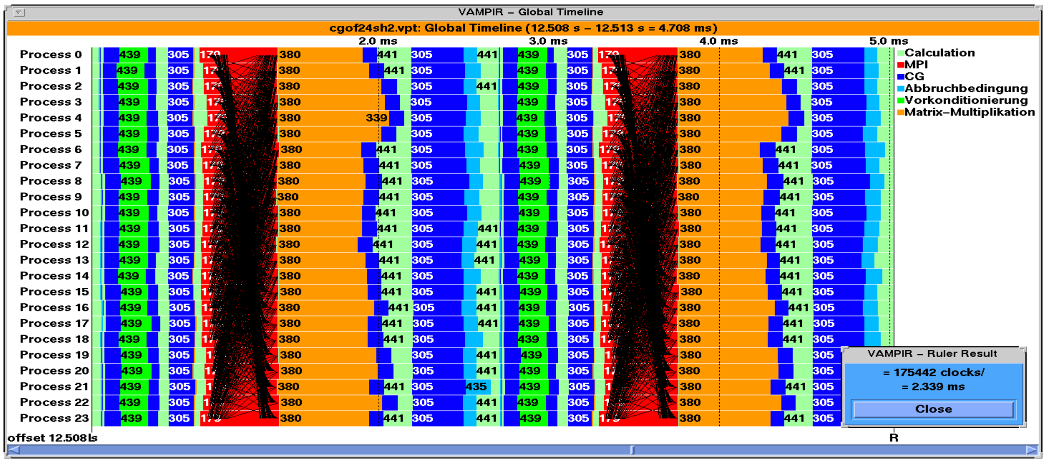


Figure 3. Two equivalent iterations in the optimized implementation

Figure 1. Together with several numerical optimizations – like storing the inverse of the diagonal of the matrix to replace divide operations by multiplications – the computing time of the linear solver could be reduced from 0.435 s to 0.234 s. Figure 2 and 3 display the timeline diagrams of two iterations on the second finest grid of the cascadic solver. On the fine grid, mainly the numerical part of the iteration is significant whereas the communication plays the most important roll on the coarser grids of the cascadic algorithm. On all grids, the average of the communication part was 40 % of the computing time in the optimized implementation.

The optimization of the communication was mainly based on the counter profiling. Additional tools – here trace-based profiling with Vampir – helped to optimize the numerical part.

5 User Feedback

The first feedback we received from our users by calling our MPI hotline to get help in optimizing their MPI programs after they noticed in the profiling e-mail that their MPI percentage was worse than expected. For a detailed feedback from our users we made a survey with 20 questions; finally, we received results from 28 of our users.[11] Major results of the survey are:

- The automatic profiling was regarded as *useful* in most answers.
- In the past, automatic profiling was only occasionally used for optimization because it came too late for most users, i.e. the application development was already completed and the programs are used in production.
- 75 % of those interviewed believe that the counter profiling can help to improve their applications in the future.
- Our weekly mail should be improved, too.
- We should provide the users with reference values. This would simplify the decision as to whether one should focus on further optimization.
- The analysis of the instruction rates of the applications' numerics and the MPI timings of the applications' communication are similarly important for the users.

For this survey, we have chosen customers who had computed at least four parallel jobs between February and April '99 and who already had received weekly mails from the automatic counter profiling system. The survey was taken by phone. Out of the 28 customers, 26 users are familiar with the automatic profiling. Out of these 26 users, 9 users always read the weekly mail, 9 users only sometimes, and 7 users rarely. Additional answers have shown that this depends on whether the users are testing their applications or, whether they have production jobs. This means that the level 1 and 2 of the profiling system is well accepted, which is also expressed by the average rank of 1.9 chosen by the 26 customers in the range of 1=*very useful* to 6=*not useful*. The third level was used by 8 users. The fourth level was used by one person. The fifth level never was used (perhaps because of the default that shows the total instruction rate and the floating point instruction rate). But the answers to the question whether instruction rates or MPI timings are more important were: *MPI timings are more important* (37 %), *both equal important* (53 %) and *instruction rates are more important* (10 %). The counter profiling is used to get information about the behavior of the users' programs (18 answers) and for optimizing the programs (11 answers). 4 users said that the profiling had really helped to improve their programs, 2 persons indicated that the MPI profiling was available too late, i.e. after they had finished the development. For the future, 21 users believe that the profiling will or may help to improve their programs. Additional optimization and analyses tools were used by

11 customers, i.e. only about 42 % have stepped to level 6. But only one user said that the MPI profiling had helped for the decision to use additional tools and 4 users told us that the profiling was available too late so that it could not help make this decision. On our system, mainly Vampir and Apprentice are used as additional optimization tools.

The last questions examined aspects of the interaction of the computing center with its customers. We also asked whether our customers – they all have publicly funded computing time on our systems – feel disturbed because the computing center also might look at the profiling information. Nobody was upset about this. Should the computing center use the overview and select optimal applications to learn about the optimization strategies used by the developers? 19 users would appreciate this, 5 don't care and 2 would be bothered. Should the computing center use the profiling information to contact the user with applications that could be inefficient? To our surprise, 22 users would like this, 2 don't care and 2 would be bothered.

Major wishes mentioned by the users are: To obtain reference values for the instruction rates, e.g. the minimum, maximum and average achieved by the other users; the layout of the weekly mail should be improved and should support html-based and ASCII-based mail readers; and it would be nice if the per-job information could be acquired from a database with a web interface.

6 Related and Future Work

Riek et al.[12] give a comprehensive overview about monitoring and profiling systems. The hardware counters are accessed by using the PCL library.[1] Trace-based systems are described in the references[2,3,7,8]. HP[4] has developed a local, user callable MPI counter profiling. Li and Zhang[5] combine counter profiling and a virtual clock approach to minimize the intrusiveness of the instrumentation. Further papers about the counter-based MPI profiling focus on the technical principles and statistical results of half a year of profiling nearly all MPI applications running on a CRAY T3E 900-512 (instrumented without the hardware counters)[9] and on the integration of the hardware counters and the implementation on different MPI library interfaces.[10] The scalability of the user interface has an analogy in the level-structured approach to learning, described in Shneiderman's[13] Principle 1.

In the future we plan to generate global half-year statistics that include hardware counters. Also, it is planned to implement a user interface, which allows to read the data on the syslog file directly via a web interface. This reduces costs in level 3 to few clicks in the web browser. Based on the technology of TOPAS[6], the hardware counter profiling can be extended from the MPI applications to *all* applications.

Acknowledgments

We gratefully acknowledge the support of the PCL library from R. Berrendorf and H. Ziegler which we have integrated into the new profiling interface. The authors especially would like to thank Roland Rühle from HLRS for placing major resources at the project's disposal.

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. 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, pp 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. K-C. Li and K. Zhang, *Supporting Scalable Performance Monitoring and Analysis of Parallel Programs*, The Journal of Supercomputing, 13, pp 5–31, 1999.
6. B. Mohr, *TOPAS – Automatic Performance Statistics Collection on the CRAY T3E*, Proceedings of the 5th European SGI/Cray MPP Workshop, Sept. 9–10, 1999. www.cineca.it/mpp-workshop/abstract/bmohr.htm and www.fz-juelich.de/zam/docs/autoren99/mohr.html
7. W.E. Nagel et al., *VAMPIR: Visualization and Analysis of MPI Resources*, Supercomputer 63, Volume XII, Number 1, Jan. 1996, pp 69–80, and technical report KFA-ZAM-IB-9528, www.fz-juelich.de/zam/docs/printable/ib/ib-95/ib-9528.ps
8. W.E. Nagel and A. Arnold, *Performance Visualization of Parallel Programs: The PARvis Environment*, technical report, Forschungszentrum Jülich, 1995. www.fz-juelich.de/zam/PT/ReDec/SoftTools/PARtools/PARvis.html
9. 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, pp 77–85. www.hlrs.de/people/rabenseifner/publ/publications.html#MPIDC99
10. R. Rabenseifner, *Automatic Profiling of MPI Applications with Hardware Performance Counters*, in J. Dongarra et al. (eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proceedings of the 6th PVM/MPI European Users' Group Meeting, EuroPVM/MPI'99, Barcelona, Spain, Sept. 26–29, 1999, LNCS 1697, pp 35–42. www.hlrs.de/people/rabenseifner/publ/publications.html
11. R. Rabenseifner, *Umfrage zum automatischen MPI Counter Profiling auf der T3E*, only online: www.hlrs.de/mpi/umfrage_results.html
12. 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
13. B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer-Interaction*, 3rd ed., Addison-Wesley, March 1998.