Chapter 7
Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir

Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf

Abstract This paper gives an overview about the Score-P performance measurement infrastructure which is being jointly developed by leading HPC performance tools groups. It motivates the advantages of the joint undertaking from both the developer and the user perspectives, and presents the design and components of the newly developed Score-P performance measurement infrastructure. Furthermore, it contains first evaluation results in comparison with existing performance tools and presents an outlook to the long-term cooperative development of the new system.

7.1 Introduction

The HPC community knows many established or experimental tools that assist programmers with the performance analysis and optimization of parallel target applications. Well known examples are HPCToolkit [1], Jumpshot [20], Paraver [12], Periscope [2], Scalasca [8], TAU [16], and Vampir [11]. They form an important contribution to the HPC ecosystem, because HPC is naturally obsessed with good performance and high efficiency, on one hand, and with more and more complex hardware and software systems, on the other hand – in this situation, sophisticated tools are an absolute necessity for analyzing and optimizing complicated parallel performance behavior.

A. Knüpfer (✉) · C. Rössel · D. an Mey · S. Biersdorff · K. Diethelm · D. Eschweiler · M. Geimer · M. Gerndt · D. Lorenz · A. Malony · W.E. Nagel · Y. Oleynik · P. Philippen · P. Saviankou · D. Schmidl · S. Shende · R. Tschüter · M. Wagner · B. Wesarg · F. Wolf
ZIH, TU Dresden, 01062 Dresden, Germany
e-mail: andreas.knuepfer@tu-dresden.de

H. Brunst et al. (eds.), *Tools for High Performance Computing 2011*,
DOI 10.1007/978-3-642-31476-6_7,

There are several established performance tools that co-exist today and many more experimental ones. They focus on different aspects and provide complementary specialized features, which makes it worthwhile to use them in combination. But at the same time, there are many similarities and overlapping functionality, in particular redundant basic functionality. This is almost necessarily so, because there is only a small number of options for several aspects, for example:

- Parallelization models (e.g. message passing, multi-threading, PGAS),
- Performance data acquisition (e.g. instrumentation, sampling, assisted by hardware components),
- Performance data representation (events, statistics).

Usually, a particular tool covers multiple combinations from this list. Therefore, one finds many redundancies in existing tools that serve the same purpose. Typical examples are source code instrumentation, profiling or event recording for MPI, and data formats. Often, one implementation could replace the other in terms of functionality and only the development histories of different tools prevent them from (re-)using the same components.

We argue that this fragmentation in the performance tools landscape brings some disadvantages for both groups concerned, the developers and the users. And we are convinced that the joint approach presented in this paper will be of mutual benefit.

7.1.1 Motivation for a Joint Measurement Infrastructure

The redundancies found in the tools landscape today bring a number of disadvantages for the developers as well as the users of the tools. From the tool developers perspective, redundancies are mostly found in the data acquisition parts, such as instrumentation, data acquisition from various sources, and in the collected data and data formats. The actual data evaluation approaches are diverse enough to justify separate tools. Thus, our goal is a joint measurement infrastructure that combines the similar components and interfaces to diverse tools. This will save redundant effort for development, maintenance, porting, scalability enhancement, support, and training in all participating groups. The freed resources will be available for enhancing analysis functionality in the individual tools.

From the user perspective, the redundancies in separated tools are a discomfort because it requires multiple learning curves for different measurement systems. Incompatible configurations, different options for equivalent features, missing features, etc. makes it harder to switch between tools or combine them in a reproducible manner. Since parallel programming and performance analysis at large scales are by no means trivial, the fragmentation in tools adds to the existing challenge for the users. The same is true for incompatible data formats that in principle transport the same data. At best, unnecessary data conversion is required, at the worst this causes repeated measurements of the same situations with different tools. Both can become

very expensive for today's highest-scale use cases. Last but not least, installation and updates of the software packages require redundant effort on the user side. A joint infrastructure will remove many of those obstacles and improve interoperability.

In the following section, the paper introduces the SILC and PRIMA projects as the frame of all presented work. The main components of the joint infrastructure are discussed in Sects. 7.3–7.7, followed by early evaluations of the run-time measurement component and the event trace data format in Sect. 7.8. The paper ends with an outlook to future goals beyond the projects' funding periods and the conclusions.

7.2 The SILC and PRIMA Projects

The SILC project[1] ("Skalierbare Infrastruktur zur Automatischen Leistungsanalyse Paralleler Codes", Engl. "Scalable Infrastructure for Automatic Performance Analysis of Parallel Codes") and the PRIMA project[2] ("Performance Refactoring of Instrumentation, Measurement, and Analysis Technologies for Petascale Computing") bring together the following established performance tools groups:

- The Scalasca groups from Forschungszentrum Jlich, Germany and the German Research School for Simulation Sciences, Aachen, Germany,
- The Vampir group at Technische Universitt Dresden, Germany,
- The Persicope group at Technische Universitt Mnchen, Germany, and
- The TAU group at University of Oregon, Eugene, USA,

and as further partners

- The Computing Center at RWTH Aachen, Germany, and
- The GNS mbH, Braunschweig, Germany, as industry partner.

All partners have a long history of mutual cooperation. The projects' main goal is to provide a joint infrastructure to address the disadvantages of separate tools as discussed above. The new infrastructure created in SILC and PRIMA has to support all central functionalities that were present in the partner's tools beforehand:

Periscope is an on-line analysis tool based on profiling information. During run-time it evaluates performance properties and tests hypotheses about typical performance problems, which are reported to the user if detected.

Scalasca performs post-mortem analysis of event traces and automatically detects performance critical situations. The results are categorized and hierarchically arranged according to types of performance problems, source code locations, and physical (computing hardware) or logical (computation domain) location.

[1]The SILC project is funded by the BMBF Germany under grant number 01IH08006.

[2]The PRIMA project is funded by the US DOE under grant number DE-SC0001621.

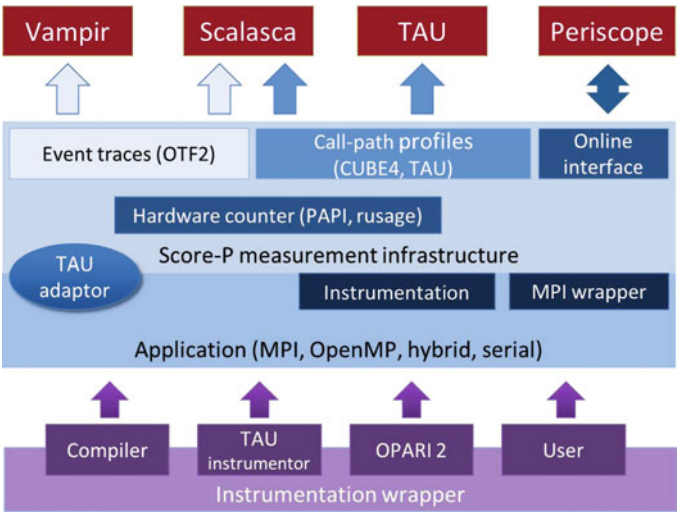

Fig. 7.1 Architecture of the Score-P instrumentation and measurement system with connection points to the currently supported analysis tools

TAU is a extensive profiling tool-set, which allows to collect various kinds of profiles such as flat profiles, phase profiles, and call-path profiles. Furthermore, it provides flexible visualization and correlation of performance data.

Vampir is an interactive event trace visualization software which allows to analyze parallel program runs with various graphical representations in a post-mortem fashion.

A number of design goals were formulated to address the needs of the four tools as well as new tools by other groups in the future. On the one hand, the primary functional requirements are:

- Support profiling and event tracing,
- Initially use direct instrumentation, later also add sampling,
- Allow off-line access (post-mortem analysis) and on-line access (live analysis),
- Initially target parallelization via MPI 2.1 and OpenMP 3.0 as well as hybrid combinations, later also include CUDA, OpenCL, and others.

On the other hand, the following non-functional requirements were considered:

- Portability to all major HPC platforms and robustness,
- Scalability to petascale level,
- Low measurement overhead,
- Easy single-step installation (through the UNITE framework),
- Open source with New BSD license.

From those goals and the experience from the existing tools, the architecture shown in Fig. 7.1 was derived. It contains the high-level components Score-P, OTF2, CUBE4, and OPARI2 which are presented in detail in the following sections.

7.3 Score-P

The Score-P component consists of an instrumentation framework, several run-time libraries, and some helper tools. The instrumentation allows users to insert measurement probes into C/C++ and Fortran codes that collect performance-related data when triggered during measurement runs. In order to collect relevant data, e.g., times, visits, communication metrics, or hardware counters, the applications need to link against one of several provided run-time libraries for serial execution, OpenMP or MPI parallelism or hybrid combinations. Extensions for POSIX threads, PGAS and CUDA are planned. The collected performance data can be stored in the OTF2, CUBE4, or TAU snapshot formats or queried via the on-line access interface.

Let's look at the instrumentor command *scorep* in more detail. It is used as a prefix to the usual compile and link commands, i.e., instead of `mpicc -c foo.c` one uses the command `scorep mpicc -c foo.c`. The instrumentor detects the programming paradigm used, MPI in this case, and adds appropriate compiler and linker flags before executing the actual build command. Currently, the following means of instrumentation are supported:

- Compiler instrumentation,
- MPI library interposition,
- OpenMP source code instrumentation using OPARI2 (see Sect. 7.6),
- Source code instrumentation via the TAU instrumentor [6], and
- User instrumentation using convenient macros.

These instrumentation variants are configurable by options to the `scorep` command. It is also possible to use the Score-P headers and libraries directly. In this case, the helper tool `scorep-config` provides all necessary information.

Once the application is instrumented, the user just needs to run it in order to collect measurement data. In contrast to the predecessor measurement systems, a flexible and efficient memory management system was implemented that stores data in thread-local chunks of pre-allocated memory. This allows an efficient distribution of the available memory to an arbitrary number of threads and minimizes the perturbation and overhead due to run-time memory allocations.

By default, Score-P runs in profiling mode and produces data in the CUBE4 format (see Sect. 7.5). This data provides first insight about the hot-spots in the application and allows to customize options for subsequent measurement runs, for example, to select MPI groups to record, to specify the total amount of memory the measurement is allowed to consume, to adapt trace output options, to specify a set of hardware counters to be recorded, and more. To keep the amount of measurement data manageable, one can include or exclude regions by name and/or wild-cards (filtering). In tracing mode, one might restrict the recording to specific executions of a region (selective tracing). In the end, the recorded data (either profiling or tracing data or both) is written to a uniquely named experiment directory. When switching from profiling to tracing or on-line access for successive experiments, there is no need to recompile or re-instrument the target application.

Furthermore, two important scalability limitations of former generations of tools have been addressed. The first affected codes using many MPI communicators, which were not handled adequately at very large scales. A new model according to [9] solved this for Score-P. The second affected the unification of identifiers at the very end of measurement. It generates globally unique identifiers from the local identifiers used during data recording. In the past, an algorithm with linear complexity with respect to the number of parallel processes was used. Obviously, this cannot match today's and future scalability demands. The new implementation uses a tree-based reduction as presented in [7]. With these fundamental improvements, Score-P is fit for the scalability levels of future supercomputer generations.

7.4 The Open Trace Format Version 2

The Open Trace Format Version 2 (OTF2) is a newly designed software package based on the experiences of the two predecessor formats OTF1 [10] and EPILOG [19], the native formats of Vampir and Scalasca, respectively.

Its main characteristics are similar to other record-based parallel event trace formats. It contains events and definitions and distributes data storage over multiple files. It uses one anchor file, one global definition file, n local definition files, and n local event files for n processes or threads. Inside each event file, the event records are stored in temporal order. The essence of OTF2 is not only the format specification but also a read/write API and a library with support for selective access. OTF2 is available as a separate software package, but also a central part of the Score-P run-time system where it acts as the memory buffer for event trace collection. It uses the same binary encoding for the memory buffers and the file representation.

As a special feature, OTF2 supports multiple I/O substrates – these are exchangeable back-ends that allow different treatment of the I/O. The standard substrate stores the OTF2 memory buffer representation in multiple files. As an alternative, a compression substrate uses ZLib for data compression of the individual files. A future extension plans to pass persistent memory pages from the trace processes or threads to post-mortem analysis processes and thus avoid I/O altogether. Last but not least, the SION library [4] substrate will address the challenge of massively parallel data and meta-data requests which overcharge parallel file systems. This is a fundamental problem on highest-scale HPC machines today. For more details about OTF2 in general see [3].

7.4.1 The SIONlib OTF2 Substrate

OTF2 distributes the records for each thread or process into separate files to avoid additional synchronization during measurement. While this is not problematic for lower scales (e.g., smaller than 1,000 ranks), it becomes a severe problem for parallel file systems at large scales. The main reason is insufficient scalability of

meta-data handling in parallel file systems. The write bandwidth is usually no limitation.

The problem with huge amounts of file handles is a widely known issue also for other applications in high-performance computing. SIONlib is a parallel I/O library which was developed at Forschungszentrum Jülich to overcome this problem by mapping virtual file handles onto a single physical file. This approach has been shown to scale up to several thousands of processes [4] and is, therefore, suitable to enhance the scalability of OTF2. In OTF2, the SION library is integrated as an additional I/O substrate which produces two separate multi-files, one for all definitions and one for all events.

7.5 The CUBE4 Format and GUI

CUBE is a profiling data model and file format. The data model describes the performance behavior of an application along three dimensions. The first dimension is a set of performance metrics. They characterize the kind of the performance issues under consideration. The second dimension is the call tree, which shows the place in the application execution where a certain issue appears. The third dimension is a description of the system which executed the application. Each triple of coordinates in this three-dimensional performance space is mapped onto a numeric value, which represents the severity of a given performance metric while visiting a given call path at a given system location.

To store measured profile data, Score-P uses the CUBE4 framework [7]. Like its predecessor CUBE3, CUBE4 provides a set of libraries and tools to store and analyze performance profiles. It consists of a writer library designed for scalability, a general-purpose C++ reader and re-writer library, a set of tools for manipulating measured profiles, and a graphical user interface for visual inspection of profiles. In contrast to its predecessor, CUBE4 also provides a Java reader library. The data format differs from CUBE3 in that the severity values, which constitute the bulk of the data, are now stored in a binary format. XML is retained only for the metadata part, since using XML for everything turned out to be too inefficient when large-scale data had to be written. GUI response times are improved using techniques such as dynamic loading, inclusive storage, and sparse representation of the data.

7.6 The OPARI2 Instrumentor for OpenMP

The source-to-source instrumentor OPARI is used to automatically wrap OpenMP constructs like parallel regions with calls to the performance monitoring interface POMP [14]. OPARI is used in many performance tools like Scalasca, Vampir and ompP [5]. In order to support version 3.0 of the OpenMP specification [15], OPARI

was enhanced to support OpenMP tasking and to provide POMP implementers with information for OpenMP nesting.

The new way to exploit parallelism in OpenMP programs using tasks posed two critical challenges for event-based analysis tools. Firstly, with OpenMP tasking, where tasks can be suspended and resumed, it is no longer guaranteed that the order of region enter and exit events follows a strict LIFO semantics per thread (the call stack). The solution to this problem is to distinguish individual task instances and to track their suspension and resumption points [13].

Secondly, with OpenMP nesting the number of threads in a parallel region is a priori unknown and the thread IDs are no longer unique, whereas performance tools traditionally rely on pre-allocated buffers for a fixed set of threads indexed by the thread IDs. Therefore, OPARI2 provides an upper bound for the number of threads used in the next parallel region as assistance for the run-time system.

In addition, OPARI2 comes with an enhanced linking procedure that allows to keep all instrumentation information within the compilation units themselves. Additional index files are no longer needed, which improves the usability of OPARI2 for multi-directory builds and pre-compiled libraries. With the presented improvements, the new major version OPARI2 addresses all aspects of state-of-the-art parallelization with OpenMP, either alone or in combination with MPI.

7.7 The On-Line Access Interface

The On-line Access interface offers a remote access path to the rich profiling capabilities of Score-P. It allows to configure measurement parameters, retrieve profile data, and to interrupt and resume the application execution. This functionality enables tools for iterative on-line performance analysis.

The On-line Access interface accepts socket connections from a remote analysis tool and communicates via the extensible and easy to use text-based Monitoring Request Interface (MRI) protocol which supports three classes of requests:

- *Monitoring requests* to select performance metrics to be recorded and to adjust measurement parameters,
- *Data retrieval requests* to fetch selected performance data, and
- *Application execution control requests* to control the measurement window based on predefined suspension points during application execution.

The main advantage of the Score-P On-line Access is that on-line analysis tools can decouple measurement from analysis, which reduces perturbation of the results. Furthermore, profile data don't need to be stored to files but can be consumed by analysis components directly. One example is the Periscope on-line analysis tool [2] which queries profile data for pre-selected phases of an iterative parallel application. Based on this, it derives hypotheses about performance problems and refines the measurement for following phases to prove or disprove them.

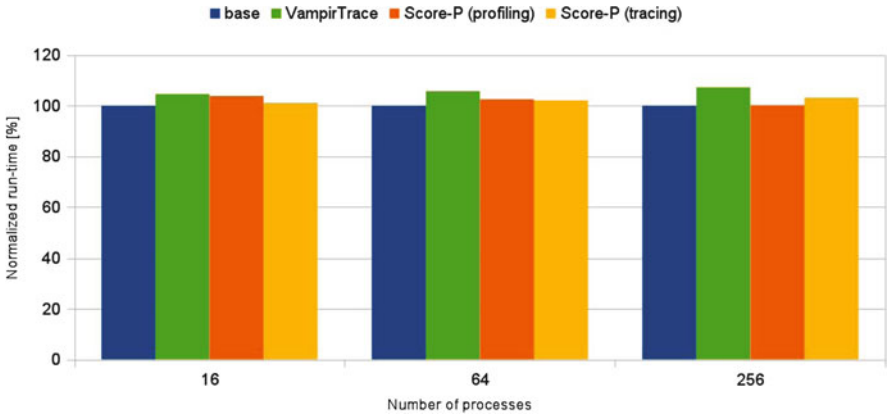

Fig. 7.2 Comparison of the run-times of the COSMO code in different modes: without instrumentation (base), with VampirTrace event tracing, with Score-P profiling, and with Score-P tracing. All times are relative to the uninstrumented run

7.8 Early Evaluation

Minimal overhead and resource consumption are essential requirements for the Score-P infrastructure. Therefore, the run-time overhead of the Score-P measurement system was compared to the existing VampirTrace infrastructure and the memory consumption of OTF2 was compared to both well-established predecessor trace formats OTF and EPILOG and the internal representation in VampirTrace.

7.8.1 Run-Time Measurement Overhead

As initial evaluation of the run-time overhead, the Score-P measurement system was tested with the COSMO code using 16, 64, and 256 MPI processes on an SGI Altix 4700. COSMO is an atmospheric model code mainly developed by the DWD[3] [17].

The baseline of this comparison are executions of the uninstrumented COSMO code. In addition, the application was executed with VampirTrace event tracing and with Score-P both in profiling and tracing mode. Figure 7.2 shows the resulting run-times. All values are normalized to the uninstrumented case and contain only

[3] Deutscher Wetterdienst (German Meteorological Service), see also http://www.cosmo-model.org/content/model/general/default.htm

the measurement phase excluding post-processing or output of measurement results to the file system. In the typical tools workflow, this indicates how close the measurement data reflects the original execution situation.

In all cases, Score-P only slightly increases the run-time compared to the unistrumented case as expected; the overhead stays below 4 %. Score-P profiling has the largest observed overhead at 16 processes (3.8 %) and the smallest one at 256 processes (0.2 %). Score-P tracing generates an overhead of 1–3.2 %. In all cases the overhead is below the overhead caused by VampirTrace (4.6–7.3 %). This indicates that Score-P is competitive in terms of overhead to VampirTrace, which has long proven its practical usefulness. All numbers show the average of at least three experiments and there were no notable outliers.

7.8.2 Trace Format Memory Consumption

Secondly, the memory consumption for event trace data is evaluated. The OTF2 data representation, which is identical to the Score-P memory buffer representation, is compared to the EPILOG format, the OTF (version 1) format, and VampirTrace's memory representation. This evaluation reveals how long the event recording can be continued with a given buffer size before it has to be interrupted for writing out data. As examples the following applications and benchmarks are used:

- The SPEC MPI2007[4] benchmarks with the test cases 104.milc ... 137.lu
- The previously mentioned COSMO code[5] from DWD
- The NAS Parallel Benchmarks[6] with nas_pb_bt (block tridiagonal solver)
- The SMG2000 Benchmark[7] (semicoarsening multigrid solver)
- The ASCI SWEEP3D benchmark[8](3D discrete neutron transport)

Figure 7.3 shows a comparison of the memory consumption of VampirTrace as the baseline, OTF, EPILOG, and OTF2. The results show that OTF2 and Score-P perform better than both predecessor tool sets and the situation behaves very similar for all test cases. OTF2 reduces the memory consumption by about 70 % compared to VampirTrace, 20–35 % compared to OTF, and 14–17 % compared to EPILOG.

[4] http://www.spec.org/mpi/

[5] http://www.cosmo-model.org/content/model/general/default.htm

[6] http://www.nas.nasa.gov/Resources/Software/npb.html

[7] https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/smg/

[8] http://www.ccs3.lanl.gov/pal/software/sweep3d/

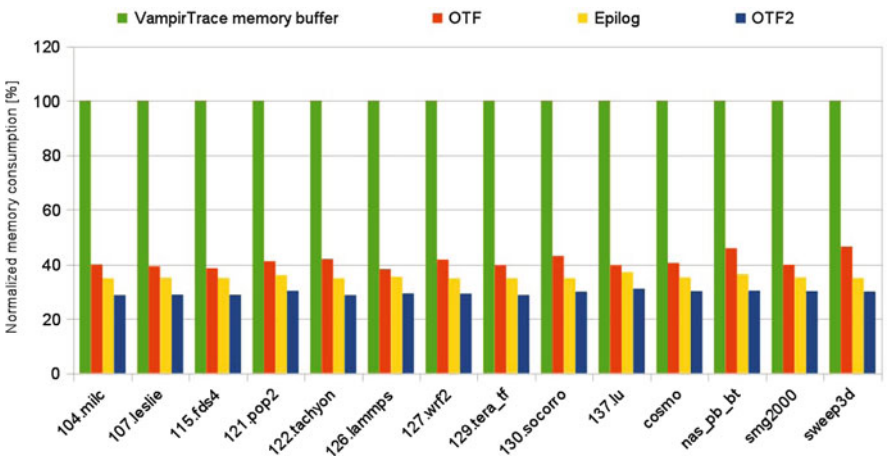

Fig. 7.3 Comparison of the event trace data memory consumption of VampirTrace, OTF, EPILOG, and OTF2 for a series of experiments with typical HPC benchmark codes

7.9 Conclusion and Outlook

With Score-P, the user community will have a single platform for large-scale performance measurements that can be interpreted with a rich collection of analysis tools. This is a first big step towards integrating the so far fragmented performance tools landscape. The primary benefits for the user are simplified installation of our tool suite, as only one measurement system has to be installed, and reduced learning effort, as the user has to read only one set of measurement instructions. Less obvious but equally important, eliminating redundancy among individual tools will free substantial tool development resources that can from now on be redirected to more powerful analysis features. At the time of writing, version 1.0 of the Score-P tool set is going to be released.

While the objective of the projects SILC and PRIMA has been the creation of an initial version, the software is already subject of a number of ongoing follow-up projects. The European project H4H (2010–2013) in the ITEA-2 framework will add extensions needed for heterogeneous architectures to Score-P with funding from the German Ministry of Education and Research (BMBF). Capabilities for compression of time-series call-path profiles [18], a feature to study the dynamic performance behavior of long-running codes, will be integrated in the EU/FP7 project HOPSA. Finally, the BMBF project LMAC will not only further refine this method but also provide functionality for the semantic compression of event traces, a prerequisite for analyzing traces of long-running applications more effectively. In addition to adding new technical features, it will also establish rules that govern Score-P's further evolution, striking a balance between robustness and stability on the one hand and innovation on the other.

References

1. Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., Tallent, N.R.: HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurr. Comput. Pract. Exp. **22**(6), 685–701 (2010). doi: 10.1002/cpe.1553. http://dx.doi.org/10.1002/cpe.1553
2. Benedict, S., Petkov, V., Gerndt, M.: PERISCOPE: an online-based distributed performance analysis tool. In: Mller, M.S., Resch, M.M., Schulz, A., Nagel, W.E. (eds.) Tools for High Performance Computing 2009, pp. 1–16. Springer, Berlin/Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-11261-4_1
3. Eschweiler, D., Wagner, M., Geimer, M., Knüpfer, A., Nagel, W.E., Wolf, F.: Open trace format 2 – the next generation of scalable trace formats and support libraries. In: Proceedings of the International Conference on Parallel Computing (ParCo), Ghent (2011). (to appear)
4. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel i/o to task-local files. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, pp. 17:1–17:11. ACM, New York, NY (2009). doi: http://doi.acm.org/10.1145/1654059.1654077. http://doi.acm.org/10.1145/1654059.1654077
5. Fürlinger, K., Gerndt, M.: ompP – a profiling tool for OpenMP. In: 1st International Workshop, IWOMP 2005, Eugene, OR, USA, June 1–4, 2005. LNCS 4315, Springer
6. Geimer, M., Shende, S.S., Malony, A.D., Wolf, F.: A generic and configurable source-code instrumentation component. In: ICCS 2009: Proceedings of the 9th International Conference on Computational Science, pp. 696–705. Springer, Berlin (2009)
7. Geimer, M., Saviankou, P., Strube, A., Szebenyi, Z., Wolf, F., Wylie, B.J.N.: Further improving the scalability of the Scalasca toolset. In: Proceedings of PARA 2010: State of the Art in Scientific and Parallel Computing, Minisymposium Scalable Tools for High Performance Computing, Reykjavik. Springer, Berlin (2010)
8. Geimer, M., Wolf, F., Wylie, B.J., Ábrahám, E., Becker, D., Mohr, B.: The scalasca performance toolset architecture. Concurr. Comput. Pract. Exp. **22**(6), 702–719 (2010). doi: 10.1002/cpe.1556
9. Geimer, M., Hermanns, M.A., Siebert, C., Wolf, F., Wylie, B.J.N.: Scaling performance tool MPI communicator management. In: Proceedings of the 18th European MPI Users' Group Meeting (EuroMPI), Santorini. Lecture Notes in Computer Science, vol. 6960, pp. 178–187. Springer, Berlin (2011). doi: 10.1007/978-3-642-24449-0_21
10. Knüpfer, A., Brendel, R., Brunst, H., Mix, H., Nagel, W.E.: Introducing the open trace format (OTF). In: Computational Science ICCS 2006: 6th International Conference, LNCS 3992. Springer, Reading (2006)
11. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The vampir performance analysis tool set. In: Resch, M., Keller, R., Himmler, V., Krammer, B., Schulz, A. (eds.) Tools for High Performance Computing, pp. 139–155. Springer, Berlin (2008)
12. Labarta, J., Gimenez, J., Martínez, E., González, P., Harald, S., Llort, G., Aguilar, X.: Scalability of tracing and visualization tools. In: Joubert, G.R., Nagel, W.E., Peters, F.J., Plata, O.G., Tirado, P., Zapata, E.L. (eds.) Parallel Computing: Current and Future Issues of High-End Computing, Proceedings of the International Conference ParCo 2005, Jülich, 13–16 Sept 2005. Department of Computer Architecture, University of Malaga, Spain, John von Neumann Institute for Computing Series, vol. 33, pp. 869–876. Central Institute for Applied Mathematics, Jülich (2005)
13. Lorenz, D., Mohr, B., Rössel, C., Schmidl, D., Wolf, F.: How to reconcile event-based performance analysis with tasking in OpenMP. In: proceedings of 6th International Workshop on OpenMP (IWOMP), Tsukuba. LNCS, vol. 6132, pp. 109–121. Springer, Berlin (2010)
14. Mohr, B., Malony, A.D., Shende, S., Wolf, F.: Design and prototype of a performance tool interface for OpenMP. J. Supercomput. **23**(1), 105–128 (2002)

15. OpenMP Architecture Review Board: OpenMP application program interface, Version 3.0. http://www.openmp.org/mp-documents/spec30.pdf
16. Shende, S., Malony, A.D.: The TAU parallel performance system, SAGE publications. Int. J. High Perform. Comput. Appl. **20**(2), 287–331 (2006)
17. Steppeler, J., Doms, G., Schttler, U., Bitzer, H.W., Gassmann, A., Damrath, U., Gregoric, G.: Meso-gamma scale forecasts using the nonhydrostatic model lm. Meteorol. Atmos. Phys. **82**, 75–96 (2003). http://dx.doi.org/10.1007/s00703-001-0592-9. 10.1007/s00703-001-0592-9
18. Szebenyi, Z., Wolf, F., Wylie, B.J.N.: Space-efficient time-series call-path profiling of parallel applications. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC09), Portland. ACM, New York (2009)
19. Wolf, F., Mohr, B.: EPILOG binary trace-data format. Technical Report FZJ-ZAM-IB-2004-06, Forschungszentrum Jülich (2004)
20. Wu, C.E., Bolmarcich, A., Snir, M., Wootton, D., Parpia, F., Chan, A., Lusk, E., Gropp, W.: From trace generation to visualization: a performance framework for distributed parallel systems. In: Proceedings of SC2000: High Performance Networking and Computing, Dallas (2000)