Score-P – A Unified Performance Measurement System for Petascale Applications

Dieter an Mey(d), Scott Biersdorf(h), Christian Bischof(d), Kai Diethelm(c), Dominic Eschweiler(a), Michael Gerndt(g), Andreas Knüpfer(f), Daniel Lorenz(a), Allen Malony(h), Wolfgang E. Nagel(f), Yury Oleynik(g), Christian Rössel(a), Pavel Saviankou(a), Dirk Schmidl(d), Sameer Shende(h), Michael Wagner(f), Bert Wesarg(f) and Felix Wolf(a,b,e)

Abstract The rapidly growing number of cores on modern supercomputers imposes scalability demands not only on applications but also on the software tools needed for their development. At the same time, increasing application and system complexity makes the optimization of parallel codes more difficult, creating a need for scalable performance-analysis technology with advanced functionality. However, delivering such an expensive technology can hardly be accomplished by single tool developers and requires higher degrees of collaboration within the HPC community. The unified performance-measurement system Score-P is a joint effort of several academic performance-tool builders, funded under the BMBF program *HPC-Software für skalierbare Parallelrechner* in the SILC project (*Skalierbare Infrastruktur zur automatischen Leistungsanalyse paralleler Codes*). It is being developed with the objective of creating a common basis for several complementary optimization tools in the service of enhanced scalability, improved interoperability, and reduced maintenance cost.

(a) Forschungszentrum Jülich GmbH, Jülich Supercomputing Centre, Leo-Brandt-Str., 52428 Jülich, Germany. e-mail: [d.eschweiler,d.lorenz,c.roessel,p.saviankou]@fz-juelich.de
(b) German Research School for Simulation Sciences, Laboratory for Parallel Programming, Schinkelstr. 2a, 52062 Aachen, Germany. e-mail: f.wolf@grs-sim.de
(c) GNS Gesellschaft für numerische Simulation mbH, Am Gaußberg 2, 38114 Braunschweig, Germany. e-mail: diethelm@gns-mbh.com
(d) RWTH Aachen University, Center for Computing and Communication, Seffenter Weg 23, 52074 Aachen, Germany. e-mail: [anmey,bischof,schmidl]@rz.rwth-aachen.de
(e) RWTH Aachen University, Dept. of Computer Science, Ahornstr. 55, 52074 Aachen, Germany
(f) Technische Universität Dresden, Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH), 01062 Dresden, Germany. e-mail: [andreas.knuepfer,wolfgang.nagel,michael.wagner2,bert.wesarg]@tu-dresden.de
(g) Technische Universität München, Fakultät für Informatik, Boltzmannstraße 3, 85748 Garching, Germany. e-mail: [gerndt,oleynik]@in.tum.de
(h) University of Oregon, Performance Research Laboratory, Eugene, OR 97403, USA. e-mail: [scottb,malony,sameer]@cs.uoregon.edu

1 Introduction

Today, computer simulations play an increasingly critical role in many areas of science and engineering, with applications growing both in number and sophistication. This creates a rising demand for computing capacity, both in terms of the number of systems and in terms of the computational power offered by individual systems. After we can no longer count on the rapid speed improvements of uniprocessors, supercomputer vendors answer this demand today with an increasing number of cores per system, forcing users to employ larger process configurations. Furthermore, modern systems feature hybrid and often also heterogeneous designs with deep memory hierarchies and advanced network architectures, further complicating the programming task. Therefore, performance-analysis tools are essential instruments in the hand of application developers that help them to cope with this complexity and to understand the performance implications of their software design choices. This is in particular true on emerging platforms whose performance characteristics are not yet well understood.

Scalability challenge. It is often neglected that many parallel programming tools face scalability challenges, just as the applications they are designed for. In fact, performance tools are most urgently needed when scaling an application to unprecedented levels, for example, in the pursuit of multi-petascale performance. Tools must even have the edge over applications with respect to the number of processes at which they can operate. In an ideal world, tools should always be readily available at the highest available scale.

Interoperability challenge. In the past, the authors developed a number of complementary performance tools such as Periscope [6], Scalasca [5], Vampir [8], and TAU [19], each focusing on a different aspects of the performance behavior. Although one would like to use them in combination, this is complicated by the fact that for historic reasons each of them uses a proprietary measurement system with its own set of data formats. Since the data formats are very similar, conversion tools alleviated this in the past. The alternative is re-running the experiment with another tool's measurement system. Both ways are very inconvenient for the users and become more troublesome with increasing scale.

Redundancy challenge. Although all four tools follow distinctive approaches and pursue individual strategies on how to address today's demand for performance-analysis solutions, they share certain features and base functionalities. This includes, for example, the instrumentation and measurement modules. Also, the data formats have very similar semantics but slightly different ways of representation. As a consequence, the growing effort required for code maintenance, feature extensions, scalability enhancements, and user support is effectively multiplied.

In this paper, we report on the status and intermediate results of the SILC project, which aims at the design and implementation of a joint measurement infrastructure for supercomputing applications called *Score-P*. The highly scalable and easy-to-use infrastructure will serve as a common basis for the above-mentioned performance tools Periscope, Scalasca, Vampir, and TAU. The project partners look back on a long history of collaboration, in particular through the *Virtual Institute – High*

Productivity Supercomputing (VI-HPS) [21], a Helmholtz-funded initiative of academic HPC tool builders from which the idea for this project emerged.

We argue that a joint performance-measurement infrastructure, the part where the overlap between the tools is significant, in combination with common data formats will not only improve interoperability but also notably reduce the overall development cost. Although a joint infrastructure will entail more coordination among developers from previously independent teams and will create more complex dependencies between the common components on the one hand and features of the individual analysis tools on the other hand, we believe that in the end, such a collaboration will save substantial resources that can be better spent on adding new features, further improving the software quality, and providing user support. For example, the savings will open the way for more powerful scalability enhancements of the measurement system alongside more advanced analysis functionality, substantially adding to the overall user value.

The next section introduces the project partners and outlines the project goals. The background and related-work section discusses the tools involved in this project as well as other well-known HPC performance analysis tools. The remainder of the paper, from Section 4 to Section 9, will discuss key software components, namely the Score-P instrumentation and runtime system, the event trace data format Open Trace Format Version 2 (OTF2), the CUBE4 profile data format, the Online Access (OA) interface, the OpenMP instrumenter Opari2, and the interface to the TAU tools. Finally, there will be an outlook on future work.

2 Project Overview and Goals

The SILC project (*Skalierbare Infrastruktur zur automatischen Leistungsanalyse paralleler Codes*, Engl. scalable infrastructure for automatic performance analysis of parallel codes) is a collaboration between the following partners:

- Center for Computing and Communication, RWTH Aachen,
- Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH), TU Dresden,
- Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH,
- Fakultät für Informatik, Technische Universität München, and
- Gesellschaft für numerische Simulation mbH (GNS), Braunschweig,

coordinated by TU Dresden. Also, the following associated partners are involved:

- Performance Research Laboratory, University of Oregon, Eugene/OR/USA,
- German Research School for Simulation Sciences, Aachen, and
- Gesellschaft für Wissens- und Technologietransfer, TU Dresden.

It was proposed by the consortium in 2008 and is funded under the BMBF call *"HPC-Software für skalierbare Parallelrechner"* from 01/2009 until 12/2011. It is carried out in close cooperation with PRIMA, a joint project between the University of Oregon and Forschungszentrum Jülich funded by the US Department of Energy.

The SILC Project Goals

The SILC project will design and implement the joint measurement infrastructure Score-P as a common basis for the performance tools Periscope, Scalasca, Vampir, and TAU. Score-P will provide the following functional requirements:

- Provide event trace recording and profile collection functionality satisfying the needs of all analysis tools involved.
- Implement direct instrumentation of target applications, as currently provided by the proprietary solutions. We plan to add sampling as an alternative in the future.
- Support postmortem and on-line analysis.
- Focus on target applications using MPI and/or OpenMP. In the future, also heterogeneous platforms with GPGPUs should be supported.
- Integrate all tools with the new infrastructure such that they provide their original functionality on top of it. The individual analysis tools will be continued as separate tools under their own names by their respective groups, however.

In addition, the partners agreed on the following non-functional requirements:

- The software should be portable to all relevant Unix based HPC platforms.
- The new infrastructure has to scale to the peta-scale level, that means to hundreds of thousands of processes or threads.
- The measurement overhead should be minimized to produce as little perturbation as possible in the recorded performance data.
- The Score-P software should reach *production quality*, that means it should be robust and well tested against all expected usage scenarios. Furthermore, we will offer appropriate user documentation, support, and training.
- The release at the end of the funding period will be under a New BSD Open Source License, which allows almost any usage.

All partners are committed to a long-term collaboration to further maintain and enhance the results of the SILC project. After the funding period, the joint effort will be open to other tools groups as new partners. The SILC web page [20] and the Score-P web page [17] provide more information. A pre-release version is already available. Future updates will be announced there.

3 Background and Related Work

Since performance analysis is an important part of today's HPC application develoepment, there are a number of tools with emphasis on different aspects. They use either sophisticated profiling techniques or rely on event trace recording.

Periscope [6] is an online performance anaylsis tool that characterizes an application's performance properties and quantifies related overheads. *Scalasca* [5] is an automatic performance analysis tool which detects a wide range of performance problems and presents the result in a concise graphical representation. It is

especially well-suited for communication and synchronization bottlenecks and is extremely scalable. *Tau* [19] is an open source performance analysis framework which mainly relies on sophisticated profile recording and evaluation methods but also supports event tracing. *Vampir* [8] is an interactive event trace browser which visualizes parallel programs with a number of displays showing different aspects of the performance behavior.

The four above-mentioned tools are the primary beneficaries of the Score-P measurement system. They will remain separate tools, but closely integrated with Score-P as their common measurement system. This is an unmatched level of integration of tools from different development teams, to the best of our knowledge.

The tools *Paraver* and *Dimemas* [11] developed by the Barcelona Supercomputing Center provide interactive event trace visualization and trace-based replay. They allow performance analysis as well as simulation of parallel run-time behaviour under altered conditions. Also, the *Jumpshot* [1] series of tools by the Argone National Laboratory and the University of Chigaco provide event trace visualization in a similar way to Vampir and Paraver. *OpenSpeedShop* [16] is a community project by the Los Alamos, Lawrence Livermore and Sandia National Laboratories, and the Krell Institute. It relies mainly on profiling and sampling but supports also event tracing. The *HPCToolkit* [13] by Rice University has similar goals but uses profile recording combined with binary analysis to obtain insights into parallel performance.

4 The Score-P Measurement System

The Score-P measurement system (see Fig. 1) enables users to instrument C/C++ or Fortran applications with probes that collect performance data when triggered during measurement runs. The data is collected as traces and/or profiles and is passed on to one or more back-ends in order to be analyzed postmortem in OTF2, CUBE4 or TAU snapshot format or by Periscope via the on-line interface.

Score-P supports the programming paradigms *serial*, *OpenMP*, *MPI* and *hybrid* (MPI combined with OpenMP). In order to instrument an application, the user needs to recompile the application using the Score-P instrumentation command, which is added as prefix to the original compile and link lines. It automatically detects the programming paradigm by parsing the original build instructions and utilizes appropriate and configurable methods of instrumentation. These are currently:

- Compiler instrumentation,
- MPI library interposition,
- OpenMP source code instrumentation using Opari2 (see Sec. 8), and
- Source code instrumentation via the TAU instrumenter [4].

Additionally, the user may instrument the code manually with convenient macros provided by Score-P. Furthermore, there are ongoing efforts to add instrumentation of executables using binary rewriting. As an alternative to to direct instrumentation, we plan to provide sampling functionality in the future.

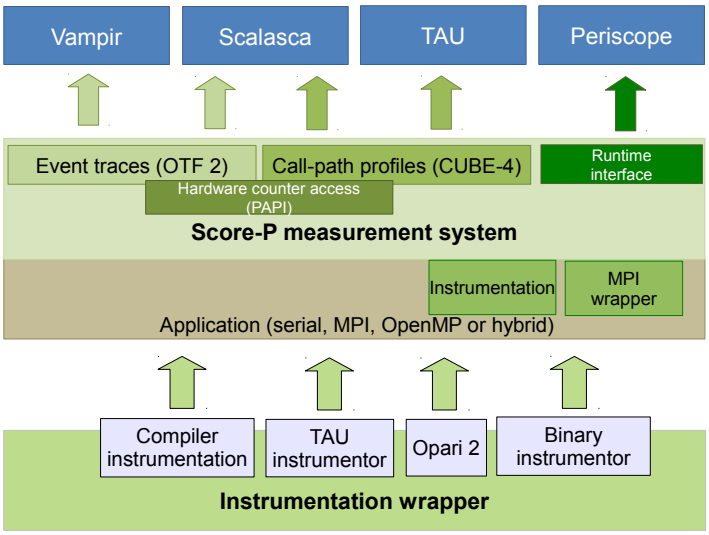

Fig. 1 Overview of the Score-P measurement system architecture and the tools interface.

During measurement, the system records several performance metrics including execution time, communication metrics, and optionally hardware counters. Performance data is stored in appropriately sized chunks of a preallocated memory buffer that are assigned to threads on demand, efficiently utilizing the available memory and avoiding measurement perturbation by flushing the data to disk prematurely.

Without recompilation, measurement runs can switch between *tracing* or *profiling* mode. In tracing mode, the performance events are passed to the OTF2 back-end (see Sec. 5) and are written to files for subsequent postmortem analysis using Scalasca or Vampir. In profiling mode, the performance events are summarized at runtime separately for each call path like in Scalasca. Additionally, we integrated support for phases, dynamic regions and parameter-based profiling known from TAU. The collected data is passed to the CUBE4 back-end (see Sec. 6) for postmortem analysis using Scalasca or TAU or is used directly through the on-line access interface by Periscope. Also in profiling mode, Score-P supports the automatic detection of MPI wait states. Usually such inefficiencies are important bottlenecks and are thoroughly investigated by means of automatic trace analysis and subsequent visual analysis using a time-line representation. In the case of Score-P wait time profiling, inefficiencies are detected immediately when the respective MPI call is completed and stored as an additional metric in the call-path profile.

5 The Open Trace Format 2

The Open Trace Format 2 (OTF2) is the joint successor of the Open Trace Format (OTF) [7] used by Vampir and the Epilog format [22] used by Scalasca. The new trace format consists of a specification of record types, in conjunction with a new trace writer and reader library. The basic OTF2 record-set is a full merge of the two predecessor formats, retaining their previous features. In the near future, OTF2 will serve as the default data sink for the upcoming Score-P measurement system (see Sec. 4), and as the default data source for the trace-analysis tools Vampir and Scalasca. This enables the user of those tools to analyze the same trace file with multiple tools without the burden of providing the same trace files in different formats. Furthermore, the user is able to combine the advantages of the different analysis tools, e.g., using Vampir to investigate the details of an inefficiency pattern that was previously detected by Scalasca.

The OTF2 library consists of three layers. The first one includes the external API and is responsible for the record representation. The record representation operates directly on the second layer, which is responsible for the memory representation of the trace data. The third layer handles the interaction with the file system and is also responsible for requests from the memory layer. The new trace library includes new features that influence its usage, which are explained below.

The external API layer comes with specifications for MPI 2.0, OpenMP 3.0 and event record types already known from OTF or Epilog. In addition, it is possible to easily add arbitrary data fields to existing records. This will simplify the process of adding new kinds of analysis data, both, for experimental additions or permanent new analysis features in the tools.

In contrast to its predecessors, OTF2 has an internal memory buffer module that is hidden from the programmer and the application using OTF2. This buffer offers several methods of reducing the size of the trace data. Thus, the tools are capable of tracing larger parts of an application without the need to interrupt and perturb the application behavior while flushing the data to disk. The two main techniques to achieve this are runtime run-length compression and support for balancing the available memory between threads of the same process (see Sec. 4).

On the lowest level, the file system interaction layer has a flexible substrate layout to support and easily add different strategies for file writing. OTF2 will support basic compressed (gzip) and uncompressed file writing, as well as more scalable approaches like writing via SIONlib [2]. Because of this flexible substrate layout, new file substrates can be easily plugged in. In addition to writing the data to disk, the complete in-memory data can be directly handed over to another application (e.g. an analysis tool). This will result in much shorter analysis cycles (time for running the measurement and examining the analysis results), because the expensive file-system operations can be skipped.

6 The CUBE Profiling Format Version 4

Designed as a generic file format for representing call-path profiles of parallel programs, CUBE is already supported by a number of HPC programming tools. These include not only Scalasca, for which it has been primarily developed, but also performance profilers like PerfSuite [10], ompP [3], TAU [19] and the MPI error detection tool Marmot [9].

A CUBE file represents summary data from a single program run. Its internal representation follows a data model consisting of three dimensions: metric, call-tree, and system. Motivated by the need to represent performance behavior on different granularity levels and to express natural hierarchical relationships among metrics, call paths, or system resources, each dimension is organized in a hierarchy. CUBE consists of a reader and writer library as well as a free graphical browser (Fig. 2) to interactively explore data files.

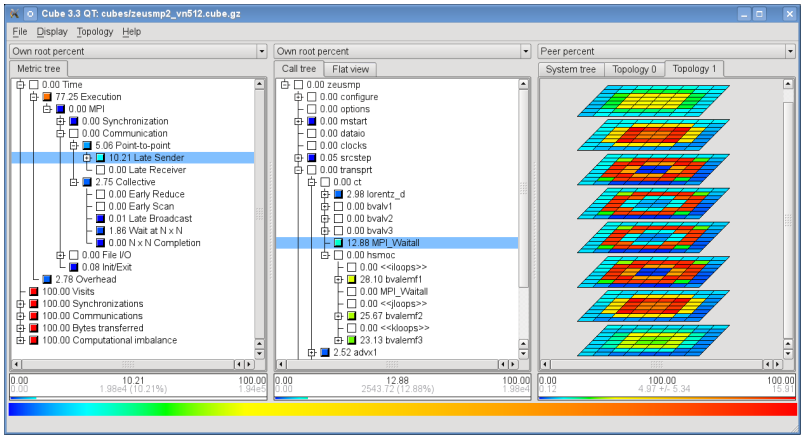

Fig. 2 The CUBE browser displaying the dimensions metric, call path, and system (left to right).

While working efficiently for applications with in the order of 10,000 processes, CUBE 3, the current version, which stores all data in a single XML file, starts reaching scalability limits beyond this scale. Major bottlenecks are writing a file to disk and the footprint of the associated memory representation when loading a file into the browser, seriously impairing interactive user experience. CUBE version 4, which is developed as part of the SILC project, will therefore introduce major changes in the service of enhanced scalability, with the two most important ones listed below:

- To speed up writing data sets, metric values will be stored in a binary format. Human-readable XML will be retained only for the metadata part.
- To reduce the memory footprint of data sets in the browser, the new format will offer random access to individual metrics, which can then be loaded separately. In addition, data for individual call paths will be stored with inclusive semantics, enabling the efficient incremental expansion of the call tree in the browser.

Finally, CUBE 4 will offer a more powerful data model, supporting the representation of time-series and parameter profiles as well as more flexibility in the specification of system-resource hierarchies and display parameters.

7 The Online Access Interface

Another novel feature of Score-P is the possibility to perform measurements in the on-line mode, i.e. control, retrieve and analyze measurements while the application is still running. There are several important benefits which are:

- Reduction of the simultaneously measured/stored performance data
- Possibility for multiple experiments within one run
- Avoiding dumping all measurements to a file at the end
- Remote analysis with measurements acquisition over networks
- Faster measurements process: one iteration of the application could be sufficient
- Monitoring configuration refinement based on already received measurements

The Score-P online access (OA) module, which is part of the measurement system, enables external agents (EA) to connect to the Score-P over TCP/IP sockets and to operate the measurement process remotely.

The part of the application execution for which performance measurements could be configured through the OA interface is called online phase. The online phase has an associated user region containing the part of application source code which is of interest for the analysis and therefore has to be marked manually by the user with the provided preprocessing directives. In order to benefit from multi-step measurements, this region should be an iteratively executed part of the code (e.g. the body of the main loop) with potential for global synchronization at the beginning and at the end. Each phase region will become a root for a call-tree profile during one measurement iteration. Data exchange with the EA takes place at the beginning and at the end of the phase, thus it does not affect the measurements within the phase.

The communication with the EA is done over TCP/IP sockets using a text-based monitoring request interface language which is a simplified subset of the request language used by Periscope. The syntax of the language covers a broad range of online analysis scenarios by means of three kinds of major requests:

- Measurement configuration request,
- Execution request,
- Measurement retrieval request.

The first category of requests allows enabling or disabling of performance metrics available in Score-P. The scope of enabled metrics is global, i.e. they are measured for every region within an online phase. Also some measurement tuning adjustments like depth limits for profile call-trees or filtering of high-overhead regions can be done with these requests. Execution requests are used to control multiple experiments by ordering Score-P to run to the beginning or to the end of the phase or, if the analysis is done, to terminate the application. Measured performance data,

stored inside the Score-P call-tree profile, can be accessed by means of measurement retrieval requests. The profile data can be returned to the EA in two ways: as a call-tree profile, where each node represents one call-path of the source code region with associated measurements attached, or as a flat profile, where measurements performed on some source code region are aggregated regardless of the call-path.

8 The Opari2 Instrumenter

Opari [14] is an established source-to-source instrumenter for OpenMP programs which is used in performance tools like Scalasca, VampirTrace and ompP. It automatically wraps OpenMP constructs like parallel regions with calls to the portable OpenMP monitoring interface POMP [14]. In order to support version 3.0 of the OpenMP specification [15], we enhanced Opari to support OpenMP tasking and to provide POMP implementors with information for OpenMP nesting. Furthermore, we improved the usability of the tool itself.

With *tasking*, the OpenMP specification introduced an additional dimension of concurrency. Although this new dimension is convenient, it challenges event-based performance analysis tools because it may disrupt the classic sequence of region entry and exit events. The solution was distinguishing individual task instances and tracking their suspension and resumption points [12].

Traditional performance analysis tools usually pre-allocate memory buffers for a fixed number of threads and store the collected data separately for each thread ID. With OpenMP *nesting* this approach needs to be adjusted because neither is the number of threads known in advance nor is the OpenMP thread ID any longer unique. Therefore, Opari2 provides an upper bound of threads in the next parallel region and an efficient mechanism to access thread-local memory.

During compilation of an application, the previous Opari approach listed all OpenMP constructs in a single file. This was inconvenient for multi-directory project layouts and it prohibited using pre-instrumented libraries or parallel builds. With the new scheme, all relevant OpenMP data stay within the instrumented compilation unit and an enhanced linking procedure offers access to the required data.

All these improvements required interface changes in the POMP specification, which justifies the step from Opari to Opari2 and from POMP to POMP2. With the new version, the established OpenMP instrumentation method is prepared for state-of-the-art parallelization with OpenMP alone or in combination with other methods.

9 Interfacing with TAU

The TAU Performance System® [19] is an open source framework and tools suite for performance instrumentation, measurement, and analysis of scalable parallel applications and systems. TAU provides robust support for observing parallel perfor-

mance (profiling and tracing) on a broad range of platforms, for managing multi-experiment performance data, and for characterizing performance properties and mining performance features.

TAU provides comprehensive performance instrumentation capabilities that support pre-processor based instrumentation implemented with the *tau_instrumentor* [4], compiler based instrumentation, MPI, POSIX I/O, CUDA, and OpenCL wrapper interposition library based on linking and pre-loading, a binary rewriter (*tau_run* based on DyninstAPI [18]), as well as Python and Java based interpreter level instrumentation implemented with JVMTI.

TAU's instrumentation interfaces with the Score-P measurement library via a special TAU adapter. TAU instrumentation can thus directly layer upon Score-P efficiently by creating a one-to-one mapping between the TAU and Score-P profiling constructs. When TAU is configured to use Score-P, it uses Score-P's MPI wrapper interposition library too. TAU's internal data structures are based on tables while Score-P has a tree based storage. This is more natural and efficient for implementing callpath profiling and further reduces TAU's measurement overhead. Using Score-P TAU can generate OTF2 traces that are unified and may be loaded in analysis tools (e.g., Vampir) without having to merge or convert trace files. The online unification of local to global event identifier also removes the need to rewrite the binary traces and the analysis stage can begin immediately after the program completes.

10 Future Work

In the remainder of the funding period, the first official release of Score-P will be prepared. This includes the completion of all scheduled features as well as quality improvements to provide a fully-functional production-quality software package under an Open Source license towards the end of 2011.

After expiration of the funding period, all partners are committed to continuing the joint development and maintenance of Score-P. This will also include user training as part of the dissemination plan. Furthermore, we plan to add new features in the mid-term future. This includes, e.g., a light-weight version of the measurement system that is suitable for permanent performance monitoring or support for accelerator architectures like GPUs with CUDA and OpenCL. Also, new analysis functionality is planned on top of the Score-P system. This may require extensions, additional data items to be collected, or online pre-processing or pre-analysis within the measurement system. Finally, at the end of the funding period, our consortium will be open to new partners who want to attach their tools to Score-P.

References

1. Chan, A., Ashton, D., Lusk, R., Gropp, W.: Jumpshot-4 Users Guide. Mathematics and Computer Science Division, Argonne National Laboratory (2007). ftp://ftp.mcs.anl.gov/pub/mpi/slog2/js4-usersguide.pdf
2. Frings, W., Wolf, F., Petkov, V.: Scalable Massively Parallel I/O to Task-Local Files. In: Proc. of the ACM/IEEE Conf. on Supercomputing, pp. 1–11 (2009)
3. Fürlinger, K., Moore, S.: OpenMP-centric Performance Analysis of Hybrid Applications. In: Proc. of the 2008 IEEE Int. Conf. on Cluster Computing, pp. 160–166. Tsukuba (2008)
4. Geimer, M., Shende, S.S., Malony, A.D., Wolf, F.: A Generic and Configurable Source-Code Instrumentation Component. In: ICCS 2009: Proc. of the 9th Int. Conf. on Computational Science, pp. 696–705. Springer, Berlin (2009)
5. Geimer, M., Wolf, F., Wylie, B.J., Ábrahám, E., Becker, D., Mohr, B.: The Scalasca Performance Toolset Architecture. Concurrency and Computation: Practice and Experience **22**(6), 702–719 (2010)
6. Gerndt, M., Fürlinger, K., Kereku, E.: Periscope: Advanced Techniques for Performance Analysis. In: Parallel Computing: Current & Future Issues of High-End Computing, Proc. of the Int. Conf. ParCo 2005, *NIC Series*, vol. 33, pp. 15–26. Forschungszentrum Jülich (2006)
7. Knüpfer, A., Brendel, R., Brunst, H., Mix, H., Nagel, W.E.: Introducing the Open Trace Format (OTF). In: Computational Science - ICCS 2006, *LNCS*, vol. 3992, pp. 526–533. Springer, Berlin (2006)
8. 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: Tools for High Performance Computing, pp. 139–155. Springer, Berlin (2008)
9. Krammer, B., Müller, M.S., Resch, M.M.: Runtime Checking of MPI Applications with MARMOT. In: Proc. of Parallel Computing (ParCo), pp. 893–900. Málaga (2005)
10. Kufrin, R.: PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux Development and Performance. In: 6th Int. Conf. on Linux Clusters: The HPC Revolution. Chapel Hill, NC (2005)
11. Labarta, J., Girona, S., Pillet, V., Cortes, T., Gregoris, L.: DiP: A Parallel Program Development Environment. In: Proc. of 2nd Int. EuroPar Conf. (EuroPar 96). Lyon (1996)
12. Lorenz, D., Mohr, B., Rössel, C., Schmidl, D., Wolf, F.: How to Reconcile Event-Based Performance Analysis with Tasking in OpenMP. In: Proc. of 6th Int. Workshop of OpenMP (IWOMP), *LNCS*, vol. 6132, pp. 109–121. Springer, Berlin (2010)
13. Mellor-Crummey, J., Fowler, R., Marin, G., Tallent, N.: HPCView: A tool for top-down analysis of node performance. J. Supercomput. **23**(1), 81–104 (2002)
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. Schulz, M., Galarowicz, J., Maghrak, D., Hachfeld, W., Montoya, D., Cranford, S.: Open|SpeedShop: An Open Source Infrastructure for Parallel Performance Analysis. Scientific Programming **16**(2-3), 105–121 (2008)
17. Score-P project page. http://www.score-p.org
18. Shende, S., Malony, A., Morris, A.: Improving the Scalability of Performance Evaluation Tools. In: Proc. of the PARA 2010 Conf. (2010)
19. Shende, S.S., Malony, A.D.: The TAU Parallel Performance System. International Journal of High Performance Computing Applications **20**(2), 287–311 (2006)
20. SILC project page. http://www.vi-hps.org/projects/silc
21. VI-HPS project page. http://www.vi-hps.org
22. Wolf, F., Mohr, B.: EPILOG Binary Trace-Data Format. Tech. rep., Forschungzentrum Jülich (2005)