Performance Refactoring of Instrumentation, Measurement, and Analysis Technologies for Petascale Computing: the PRIMA Project

Joint Final Report.
Date Submitted: January 31, 2014.

Allen D. Malony
Department of Computer and Information Science
University of Oregon
Eugene, Oregon 97403

Felix Wolf
Forschungszentrum Jülich GmbH
Wilhelm-Johnen-Strasse
Jülich 52428
Germany

University of Oregon	
Award no.:	DE-SC0001775
PI:	Prof. Allen D. Malony
Co-PI:	Dr. Sameer Shende
Grant admin.:	Charlotte Wise 5294 University of Oregon Eugene, OR 97403

Forschungszentrum Jülich GmbH	
Award no.:	DE-SC0001621
PI:	Prof. Felix Wolf
Co-PI:	Dr. Bernd Mohr
Grant admin.:	Volker Marx Wilhelm-Johnen-Strasse Jülich 52428 Germany

1 Introduction

The growing number of cores provided by today's high-end computing systems present substantial challenges to application developers in their pursuit of parallel efficiency. To find the most effective optimization strategy, application developers need insight into the runtime behavior of their code. The University of Oregon (UO) and the Jülich Supercomputing Centre of Forschungszentrum Jülich (FZJ) developed the performance analysis tools *TAU* [23] and *Scalasca* [12], respectively, which allow high-performance computing (HPC) users to collect and analyze relevant performance data – even at very large scales. TAU and Scalasca are considered among the most advanced parallel performance systems available, and are used extensively across HPC centers in the U.S., Germany, and around the world.

The TAU and Scalasca groups share a heritage of parallel performance tool research and partnership throughout the last fifteen years. Indeed, the close interactions of the two groups resulted in a cross-fertilization of tool ideas and technologies that pushed TAU and Scalasca to what they are today. It also produced two performance systems with an increasing degree of functional overlap.

While each tool has its specific analysis focus, the tools were implementing measurement infrastructures that were substantially similar. Because each tool provides complementary performance analysis, sharing of measurement results is valuable to provide the user with more facets to understand performance behavior. However, each measurement system was producing performance data in different formats, requiring data interoperability tools to be created. A common measurement and instrumentation system was needed to more closely integrate TAU and Scalasca and to avoid the duplication of development and maintenance effort.

The *PRIMA* (Performance Refactoring of Instrumentation, Measurement, and Analysis) project was proposed over three years ago as a joint international effort between UO and FZJ to accomplish these objectives:

- refactor TAU and Scalasca performance system components for core code sharing
- integrate TAU and Scalasca functionality through data interfaces, formats, and utilities

As presented in the following report, the project has completed these goals. In addition to shared technical advances, the groups have worked to engage with users through application performance engineering and tools training. In this regard, the project benefits from the close interactions the teams have with national laboratories in the United States and Germany. We have also sought to enhance our interactions through joint tutorials and outreach. UO has become a member of the Virtual Institute of High-Productivity Supercomputing (VI-HPS) [28] established by the Helmholtz Association of German Research Centres as a center of excellence, focusing on HPC tools for diagnosing programming errors and optimizing performance. UO and FZJ have conducted several VI-HPS training activities together within the past three years.

The long-range direction for PRIMA was towards a robust and shared measurement infrastructure upon which will be layered interoperable parallel performance analysis capabilities. We believe this was an important and necessary goal in order to realize productive performance tools for the next-generation HPC systems. Both groups were committed to this purpose. Indeed, during the course of the PRIMA project thus far, we have joined forces with the *SILC* project and its successor the *LMAC* project, both funded by the German Ministry of Education and Research. The original objective of SILC is the creation of a common measurement infrastructure called *Score-P* [5] to support the performance tools *Periscope* [13], *Scalasca*, and *Vampir* [20] developed in Germany. Funded partners in SILC and LMAC are the Technical University of Dresden, Jülich Supercomputing Centre, the Technical University of Munich, RWTH Aachen University, and GNS GmbH, a company specializing in finite-element software services. Not to miss this opportunity for a true community-developed measurement infrastructure, we decided to carry out the integration of TAU and Scalasca (the main objective of PRIMA) in the framework of Score-P – at least at the levels of instrumentation and measurement. Having internationally leading performance-tool developers as stakeholders, Score-P is guaranteed to have a wide impact in the field. An overview of Score-P measurement is described in Section 3. While the Score-P measurement system will replace the exiting Scalasca measurement system completely, TAU will provide a backward-compatible interface.

The DOE investment in the PRIMA project has fundamentally guided the directions with Score-P. Furthermore, the PRIMA results will now have a translation path from the reference TAU/Scalasca refactoring and integration platform to a next-generation infrastructure. For instance, as common output formats, Score-P will use the newly-developed CUBE-4 format for profiles and the Open Trace Format 2 (OTF2) format for event traces. This will enable the analysis and visualization tools in the ensemble to share as much information as possible. Thus, performance data can be shared between Scalasca and TAU, but also with Periscope and Vampir. In general, we believe the direction we are pursuing in PRIMA will result in a robust measurement foundation

upon which we can build new sophisticated performance analysis capabilities for the DOE mission applications.

This report covers the whole PRIMA project. After this introduction, the report shows a brief overview over the milestones and the results that have been achieved. in Section 2. Most of the work is covered by the new instrumentation and measurement framework Score-P, which we introduce afterwards in Section 3. After the Score-P overview, we present our work in instrumentation (Section 4), measurement (Section 5), and analysis (Section 6), similar to the proposal structure in more detail. Lastly, Section 7 provides an enumeration of our dissemination efforts through publications, talks, tutorials, and meetings.

2 Milestone overview

The PRIMA proposal was structured in three work packages iPRIMA, mMPRIMA, and aPRIMA. The iPRIMA work package contains the milestones for application instrumentation. The mPRIMA work package focuses on the measurement system and the aPRIMA work package describes the goals for the performance data analysis. This section presents an overview over the status of the milestones for every work package in the proposal.

iPRIMA

Year 1	
Specify unified event model	**Done.** Contributed as part of the Score-P specification.
Specify unified instrumentation API	**Done.** Instrumentation tools now target this API. Contributed as part of the Score-P specification.
Retrofit Scalasca compiler adapters to work with TAU	**Done.** The adapters were fitted to the Score-P measurement system. Because the TAU system is interoperable with Score-P, the goal of the milestone is met.
Design of unified selective instrumentation framework	**Done.** Approaches for TAU and Scalasca selective instrumentation combined. Contributed as part of the Score-P specification.
Investigation of instrumentation with PIN	**Done.** A decision was made to build on Dyninst API and develop a binary instrumenter (Cobi) that incorporates static analysis information. During the project runtime, we started cooperation with the MAQAO developers, and changed our strategy to use the MAQAO binary instrumenter.
Year 2	
Implement joint source-code instrumentation infrastructure	**Done.** Source instrumentation utilizes TAU PDT technology.
Implement joint set of compiler adapters	**Done.** Contributed as part of Score-P.
Retarget TAU general wrapper generator to PRIMA instrumentation	**Done** as part of integrating TAU with Score-P.
Design unified configuration scripts	**Done.** Results from using the common measurement system Score-P.
Design joint binary/dynamic instrumentation infrastructure	**Done.** Contributed as part of Score-P.
Year 3	
Implement unified configuration scripts	**Done.** Results from using the common measurement system Score-P.
Implement joint binary/dynamic instrumentation infrastructure	**Done** with the implementation of Cobi and its integration in Score-P. Furthermore, binary instrumentation is possible with TAU. Because of the new cooperation with MAQAO, we will use the MAQAO binary instrumenter in the future.
Test of PRIMA instrumentation on petascale applications	**Done.** See Section 4.2.
Release full PRIMA instrumentation framework	**Done.** The Score-P was first released in January 2012. The current release version is 1.2.2

mPRIMA

Year 1	
Implement STL-style base data structures in C (phase 1)	**Done.** This was necessary to reduce C++ build dependencies in Score-P.
Define event management core	**Done.** Contributed as part of the Score-P specification.
Specify unified profiling format	**Done.** Defined Cube 4 format.
Design event unification approach	**Done.** Contributed as part of the Score-P specification.
Evaluate configuration and installation strategy	**Done.**
Year 2	
Implement STL-style base structures (phase 2)	**Done.** Included with Score-P.
Implement event management core	**Done** as part of Score-P.
Design integrated profiling framework and scalable output algorithm	**Done.** Included as part of Score-P.
Implement configuration and installation approach	**Done.** Included as part of Score-P.
Develop event unification methods	**Done.** Included as part of Score-P.
Year 3	
Implement scalable profile output algorithm	Tests showed no improvents. See Section 6.1.1.
Develop call-path profiling based on Scalasca tree structure	**Done** as part of Score-P.
Test of PRIMA measurement on petascale applications	**Done.** See Section 4.2 and Section 5.4.
Release full PRIMA measurement framework	**Done.** The first Score-P release happened in January 2012. The current version is 1.2.2.

aPRIMA

Year 1	
Design of PRIMA data model	**Done** based on TAU and Scalasca requirements. Led to new data formats.
Evaluate approaches for developing analysis interfaces	**Done.**
Design PRIMA analysis scripting	**Done.** Based on Eclipse.
Better integrate existing TAU and Scalasca parallel profiling tools	**Done.** Integration based on new profile formats.
Year 2	
PerfDMF evolved to use PRIMA data model	**Accomplished** as part of integrating TAU with Score-P.
Specify interfaces for analysis tool interoperation	**Done** for TAU ParaProf and Scalasca CUBE viewer.
Implement analysis scripting for PRIMA workflow	**Done.** Built with ETFw framework.
Design PRIMA workflow framework and identify analysis workflows	**Done.** The ETFw framework is implemented.
Year 3	
Analysis tool fully functional with PRIMA data model	**Done.** Scalasca and TAU are able to work with the new data formats.
Develop interfaces and demonstrate interoperable analysis tools	**Done.**
Develop performance analysis workflows	**Done.**
Test PRIMA analysis on petascale applications	**Done.** See Section 5.4.
Release full PRIMA analysis framework	**Done.** Scalasca 2.0 was released in August 2013.

3 The Score-P infrastructure

The Score-P software [5, 14] provides an infrastructure for instrumenting applications and for recording performance data either as profiles or event traces. The results of the instrumentation work package (iPRIMA) and the measurement work package (mPRIMA) are being integrated into Score-P, leveraging base functionality of Score-P such as a very sophisticated build system. The general architecture of Score-P is depicted in Figure 1.

The Score-P software was first released in January 2012. The current Score-P release version is 1.2.2 which was released in September 2013. In the following sections we first give an overview over the status of the instrumentation and and measurement parts in Score-P in Section 3.1 and Section 3.2.

3.1 Instrumentation with Score-P

The main goal of the iPRIMA work package is to make the instrumentation mechanisms of Scalasca and TAU interoperable. Because the Score-P measurement system can be used by both tools, the prior instrumentation mechanisms from Scalasca and TAU are being made available to Score-P and

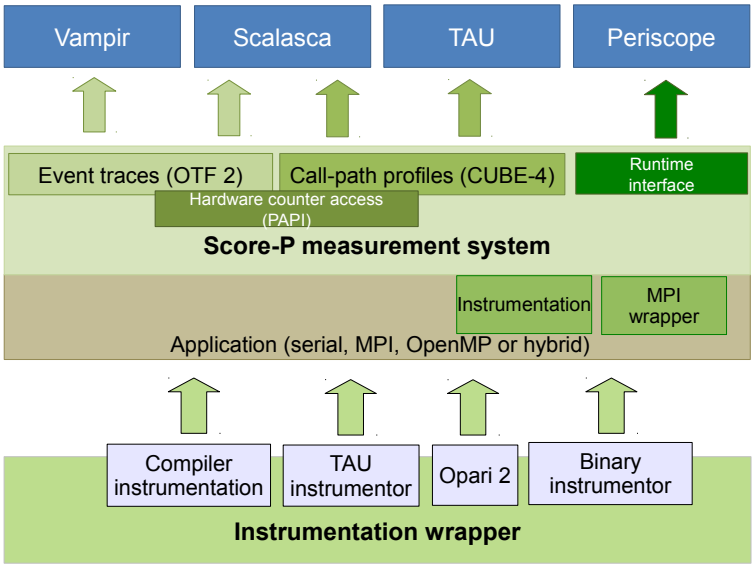


Figure 1: Overview of the Score-P measurement system architecture and the interfaces to various analysis tools.

thus, will become accessible to both tools. Currently, Score-P supports the following instrumentation techniques:

- Compiler instrumentation: The compiler adapters for the Cray, GNU, Intel, IBM XL series, PGI, and Oracle Studio compilers are completed. Other compilers that share the instrumentation interface with one of the above-mentioned compilers (e.g. the Pathscale compiler) can be used, too.
- OPARI2: OPARI2 is an automatic source code instrumentor with the main focus on OpenMP constructs. As part of the Score-P software, its predecessor OPARI [18] was improved to remove some of its limitations so that it can work with pre-instrumented libraries and source files which are distributed over several directories. OPARI2 was finished in April 2011. The Score-P adapter for the instrumentation interface of OPARI2, called POMP2, is implemented.
- TAU instrumentor: A configuration specification for the generic TAU instrumentor [11] was developed. The instrumentor is an optional feature that can be enabled if it is installed on the system.
- MPI library interposition: An appropriate wrapper for MPI exists which covers support for point-to-point and collective communication. However, one-sided communication information are not yet recorded.
- Manual user instrumentation: A set of macros are provided, which the user can insert manually into the source code. They allow the user to instrument the enter and exit points of user-defined code blocks (regions). Furthermore, parameter values can be provided in an extra statement, allowing call-path splits based on different parameter values. Other features are phase profiling and dynamic regions.
- Score-P provides an interface for external tools to input events into Score-P. This interface is used by TAU, to build a backward compatible instrumentation layer for TAU.

- Binary instrumentation: We developed the configurable binary rewriter Cobi [19], as a prototype to perform function instrumentation. Using Cobi is an optional feature until Score-P 1.2 In Score-P version 1.3 we plan to replace Cobi with the binary instrumenter from the MAQAO [17] project. We have also developed tau_rewrite, a binary instrumenter based on the MAQAO [17] project and have tested it with Score-P. Thus, switching to MAQAO unifies development in the binary instrumentation area between Score-P and TAU.

- Access to CUDA events via the CUPTI interface. This feature was developed during the reported period, but not as part of the PRIMA project.

To instrument an application, the user must rebuild the application. An instrumentation wrapper tool, which is placed in front of the original build command, takes care of the additional parameters, libraries and build steps. It tries to determine the used paradigm and chooses a reasonable set of default instrumentation techniques. E.g. OPARI2 instrumentation is only applied by default if the instrumenter detects that the application uses OpenMP and, thus, instrumentation of OpenMP directives is required. However, the user can override the default choice. The additional parameters for every build step can also be obtained from a configuration tool that outputs the appropriate commands or parameters. The usage of a common instrumentation and measurement infrastructure implies that the configuration settings of Score-P are usable for both TAU and Scalasca.

Instrumentation layer was tested with benchmarks and multiple real world applications, e.g. the BOTS benchmark suite [9], COSMO, FIRE [8], INDEED, PEPC, PFLOTRAN, Sweep3D, and XNS. We found that the Score-P software scales well.

3.2 The Score-P measurement system

The measurement system supports profiling and event tracing. If tracing is enabled, the events are passed to the OTF2 backend. If profiling is enabled, the performance metric statistics are recorded for every call path. After the measurement run, the profile can be stored either in the Cube 4 format or in the TAU snapshot format. The choice between profiling and tracing can be specified via environment variables.

For the internal representation of the profile, Score-P uses the call-tree structure of Scalasca. Currently, Score-P records the inclusive metrics runtime and visits separately for each call path and thread. Additional statistics recorded per call path for each metric include the sum, the minimum, the maximum, the sum of square values, and the number of values. The profiling backend supports the features parameter-based profiling, dynamic regions, phase profiling, tasks, and clustering of dynamic regions.

Score-P supports the programming paradigms serial, OpenMP, MPI, and hybrid (MPI combined with OpenMP). An improvement compared to the existing measurement systems is that Score-P supports nested OpenMP parallel regions and a varying number of threads. To avoid expensive locking on every event which would inhibit scalability, the measurement system contains its own memory management system. Thus, every thread operates on a separate piece of memory. Furthermore, we added support for GPU locations in the measurement system. Score-P supports PAPI [7] counters and user defined metrics

The basic measurement system is released and tested with benchmarks and real world applications. We think that the Score-P software scales well. For more details concerning the scaling, see Section 5.4.

Figure 2 shows a profile of the Mantevo [21] HPCCG miniapp instrumented using TAU's source instrumentor using PDT with support for automatic loop level instrumentation with the TAU

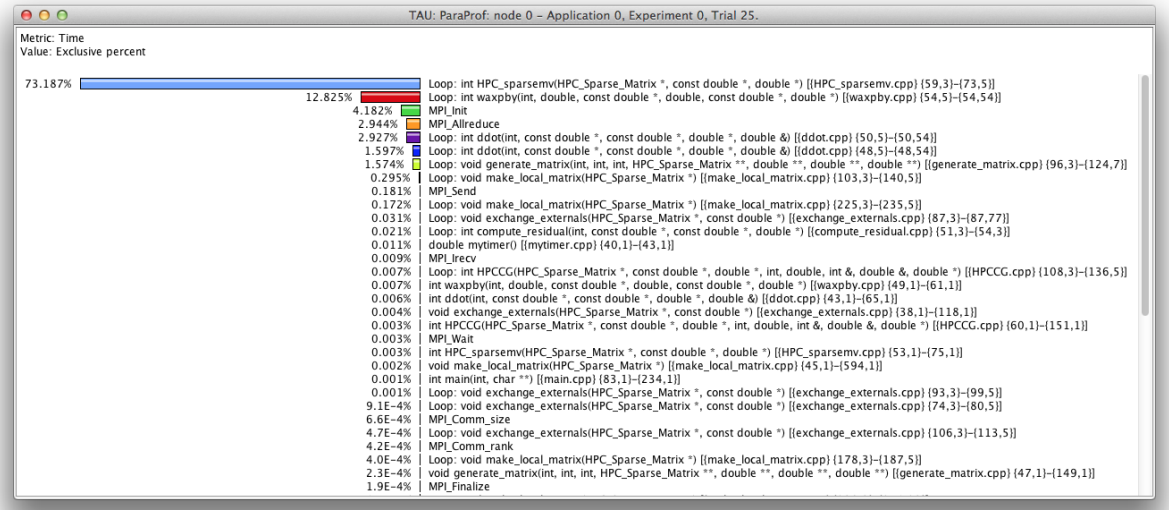


Figure 2: TAU's ParaProf profile browser shows a Cube 4 profile showing the time spent in HPCCG loops

adapter in Score-P to measure the wallclock time spent in its code regions. TAU's ParaProf profile browser reads the Cube 4 output generated by Score-P.

Figure 3 shows the profile for HPCCG at the routine boundaries using compiler-based instrumentation with the TAU adapter in Score-P. Expanding nodes in this ParaProf tree table reveals the structure of the callgraph of the miniapp. Nodes are color-coded based on the inclusive time (collapsed) or exclusive time (expanded nodes) spent in the routine.

When tracing is enabled, OTF2 traces are generated by Score-P without the need for rewriting binary event traces during the unification phase. This allows for a scalable trace generation and visualization using Score-P. Vampir reads the OTF2 trace as shown in Figure 4. This display shows the timeline view of the HPCCG miniapp instrumented with loop level instrumentation using TAU, PDT, and Score-P.

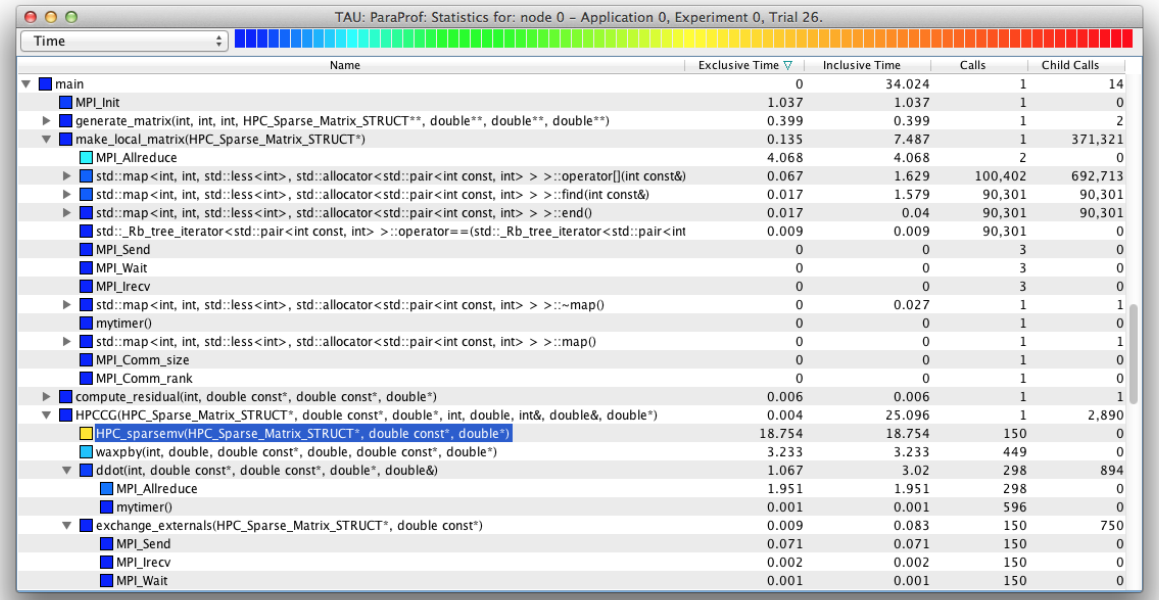
4 iPRIMA

The main goal of the iPRIMA work package is to make the instrumentation mechanisms of Scalasca and TAU interoperable. Because the Score-P measurement system can be used by both tools, the prior instrumentation mechanisms from Scalasca and TAU are being made available to Score-P and thus, will become accessible to both tools. In addition, new instrumentation capabilities continue to be developed in the context of the TAU system that accommodate new requirements and will eventually flow into the Score-P framework

4.1 Score-P Instrumentation Integration

Currently, those instrumentation techniques that have been integrated with Score-P include:

- Compiler instrumentation: Both TAU and Scalasca projects are experienced with working with different compilers to generated instrumentation. This experience was combined to develop



Name	Exclusive Time	Inclusive Time	Calls	Child Calls
main	0	34.024	1	14
MPI_Init	1.037	1.037	1	0
generate_matrix(int, int, int, HPC_Sparse_Matrix_STRUCT**, double**, double**, double**)	0.399	0.399	1	2
make_local_matrix(HPC_Sparse_Matrix_STRUCT*)	0.135	7.487	1	371,321
MPI_Allreduce	4.068	4.068	2	0
std::map<int, int, std::less<int>, std::allocator<std::pair<int const, int> > >::operator[](int const&)	0.067	1.629	100,402	692,713
std::map<int, int, std::less<int>, std::allocator<std::pair<int const, int> > >::find(int const&)	0.017	1.579	90,301	90,301
std::map<int, int, std::less<int>, std::allocator<std::pair<int const, int> > >::end()	0.017	0.04	90,301	90,301
std::_Rb_tree_iterator<std::pair<int const, int> >::operator==(std::_Rb_tree_iterator<std::pair<int	0.009	0.009	90,301	0
MPI_Send	0	0	3	0
MPI_Wait	0	0	3	0
MPI_Irecv	0	0	3	0
std::map<int, int, std::less<int>, std::allocator<std::pair<int const, int> > >::~map()	0	0.027	1	1
mytimer()	0	0	1	0
std::map<int, int, std::less<int>, std::allocator<std::pair<int const, int> > >::map()	0	0	1	1
MPI_Comm_size	0	0	1	0
MPI_Comm_rank	0	0	1	0
compute_residual(int, double const*, double const*, double*)	0.006	0.006	1	1
HPCCG(HPC_Sparse_Matrix_STRUCT*, double const*, double*, int, double, int&, double&, double*)	0.004	25.096	1	2,890
HPC_sparsemv(HPC_Sparse_Matrix_STRUCT*, double const*, double*)	18.754	18.754	150	0
waxpby(int, double, double const*, double, double const*, double*)	3.233	3.233	449	0
ddot(int, double const*, double const*, double*, double&)	1.067	3.02	298	894
MPI_Allreduce	1.951	1.951	298	0
mytimer()	0.001	0.001	596	0
exchange_externals(HPC_Sparse_Matrix_STRUCT*, double const*)	0.009	0.083	150	750
MPI_Send	0.071	0.071	150	0
MPI_Irecv	0.002	0.002	150	0
MPI_Wait	0.001	0.001	150	0

Figure 3: TAU's ParaProf tree table shows the callgraph on node 0

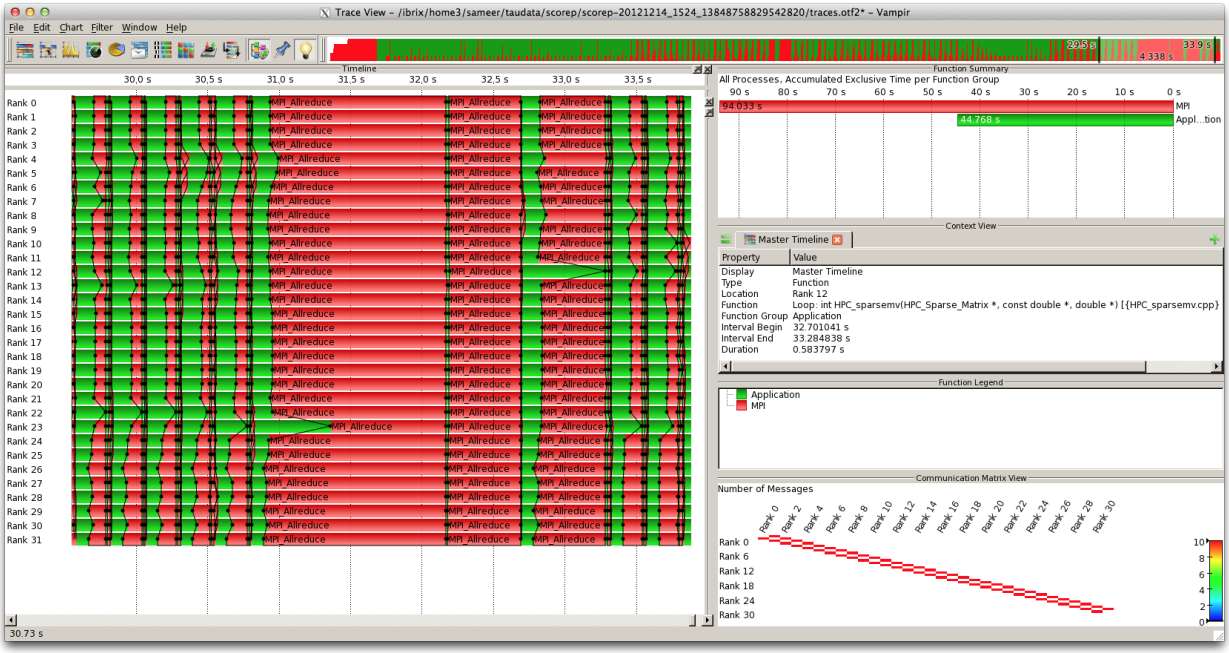


Figure 4: Vampir displays the OTF2 trace for the HPCCG miniapp

Score-P compiler adapters for the Cray, GNU, Intel, IBM XL series, PGI, and ORACLE Studio compilers. Other compilers that share the instrumentation interface with one of the above-mentioned compilers (e.g. the Pathscale compiler) can be used, too.

- OpenMP instrumentation: *OPARI* [18] is an automatic source code instrumentor with the main focus on OpenMP constructs. TAU has been a long-time consumer of OPARI to enable OpenMP instrumentation for TAU measurement. In April 2011, *OPARI2* was completed. It removes limitations of working with pre-instrumented libraries and source files which are distributed over several directories. Score-P implements an adapter for the OPARI2 instrumentation interface, called *POMP2*.

- MPI library interposition: Both TAU and Scalasca provide robust support for MPI library instrumentation through interposition. Creation of a wrapper library for MPI with has been comleted for point-to-point and collective communication. The support for one-sided communication information is currently under development by another project.

- Manual user instrumentation: A set of macros are provided, which the user can insert manually into the source code. They allow the user to instrument the enter and exit points of user-defined code blocks (regions). Furthermore, parameter values can be provided in an extra statement, allowing call-path splits based on different parameter values. Other features are phase profiling and dynamic regions.

- TAU instrumentor: The instrumenter can utilize the generic TAU instrumentor [11] for source code instrumentation of function enter/exits. Therefor, a configuration specification for the TAU instrumentor was developed jointly by the TAU and Scalasca teams and has been demonstrated with Score-P measurement. The TAU instrumentor provides much more functionality than function enter/exit instrumentation. However, to harness the full functionality the user has to invoke the TAU instrumenter with a custom instrumentation specification.

- Score-P provides an interface for external tools to input events into Score-P. This interface is used by TAU, to build a backward compatible instrumentation layer for TAU.

- Binary instrumentation: We developed the configurable binary rewriter Cobi [19], as a prototype to perform function instrumentation. Using Cobi is an optional feature until Score-P 1.2 In Score-P version 1.3 we plan to replace Cobi with the binary instrumenter from the MAQAO [17] project. We have also developed tau_rewrite, a binary instrumenter based on the MAQAO [17] project and have tested it with Score-P. Thus, switching to MAQAO unifies development for binary instrumentation between Score-P and TAU.

- Access to CUDA events via the CUPTI interface. This feature was developed during the reported period, but not as part of the PRIMA project.

To instrument an application, the user must rebuild the application. The Score-P framework is developing an instrumentation wrapper tool, which is placed in front of the original build command and will take care of the additional parameters, libraries, and build steps. The instrumentation wrapper supports compiler instrumentation, OPARI2, the TAU instrumentor, MPI library interposition, manual user instrumentation, and binary instrumentation. However, all additional information can be obtained from a configuration tool that outputs the appropriate commands or parameters. The usage of a common instrumentation and measurement infrastructure implies that the configuration settings of Score-P are usable for both TAU and Scalasca.

4.2 Testing

To ensure the functionality and robustness of our new software, we performed multiple tests.

We deployed an automatic test system which checks out the most recent version of Score-P, build it, install it and run a test suite. These tests are automatically triggered after every commit to give an immediate response on any modification that breaks our code. To cover a broad variety of platforms and compilers, these tests are performed in 20 different configurations on a dedicated test system. Furthermore, we frequently run the automatic test suite on high performance systems in more than 20 configurations. If any of the tests fails on any platform the developers are notified automatically. Furthermore, the status is displayed in the project's developer Wiki. Thus, we ensure our software is portable to our target platforms.

Currently, the test suite contains 84 tests for various components like memory management, filtering, dynamic regions, or OPARI2 instrumentation. Furthermore, it performs some builds, e.g. to check whether header files are self-contained. Furthermore, it automatically instruments a jacobi code with all possible combinations of instrumentation mechanisms and supported parallelization techniques that are supported on the particular platform. The minimal configuration contains already more than 100 possible combinations. If optional features like PDT instrumentation or binary instrumentation are enabled, the number of combinations is even larger.

Beside the automatic tests suite, Score-P was manually tested with a couple of benchmarks and real world applications:

- PEPC-P [3] is a multi-particle simulation which uses MPI and is written in Fortran. It was instrumented with the default Score-P instrumentation mechanisms and run on the BG/P system Jugene.
- PFLOTRAN [4] is a scientific code for modeling multiscale-multiphase-multicomponent subsurface reactive flows. Its excessive creation of MPI communication contexts, is a good stress test for the MPI system in Score-P. It was also tested on Jugene.
- COSMO is a wether forecase code which uses MPI. It is mainly written in Fortran but contains also some C code.
- INDEED [1] is a hybrid (OpenMP plus MPI) closed source code for metal forming simulations.
- Sweep3d is a Fortran code which we used for scaling tests because the code itself scales well to peta-scale. It was tested on Jugene and on the BG/Q system Juqueen.
- The SPEC MPI 2007 benchmark suite was instrumented for Score-P using the binary instrumenter Cobi. The results are published at the Europar Conference 2011 [19]. The test was performed on the Juropa system.
- The Barcelona OpenMP Tasking Suite (BOTS) [9] provides 8 codes using OpenMP tasking. Some of them create more than a billion of tasks, putting the newly developed tasking infrastructure to a stress test. The results are published for tracing [22] and profiling [16]. The published tests were performed on Juropa. However, we also ran the tests successfully on Judge and a Linux desktop system.
- For the Flexible Image Retrieval Engine (FIRE) [8] exists a version that uses OpenMP tasks and served as a real world test case for an OpenMP code with tasking. It is written in C++.

Furthermore, we conducted two tutorials (see Section 7) for Score-P where users brought their own codes and, thus, successfully applied on a number of further codes.

5 mPRIMA

The mPRIMA work package is the primary challenge for the PRIMA project since it will become the key core technology for TAU and Scalasca integration. Meanwhile, the common Score-P measurement system has prelaced the Scalasca measurement system and also TAU works with the Score-P measurement system.

The measurement systems of TAU and Scalasca both provided parallel profiling and tracing capabilities. TAU supported multiple multi-threading and distributed memory programming models. Scalasca supported only OpenMP, MPI, and hybrid (MPI combined with OpenMP). While TAU had its own legacy tracing library, it was possible to select either the Vampir or Scalasca tracing library. In the case of parallel profiling, the implementations in the two tools were significantly different.

5.1 Score-P Measurement Integration

Initially, Score-P targeted four programming paradigms: serial, OpenMP, MPI, and hybrid. But unlike the former Scalasca measurement system, the Score-P measurement system abstracts from the paradigms and encapsulates paradigm specific code in separate libraries. The Score-P instrumenter selects the appropriate libraries for the threading paradigm and multi-process paradigm and links only the libraries for the selected paradigms into the application. Currently, Score-P has support for the threading paradigms OpenMP and single threaded, and for the multi-process paradigms MPI and single process. In addition, support for GPUs is available. Work in progress is support for Pthreads.

An further improvement compared to the Scalasca measurement systems is that Score-P supports nested OpenMP parallel regions and a varying number of threads and tasks. The tasking support was a topic which required research and development of new concepts which we describe in further detail in Section 5.3.

The inclusion of an OTF-2 library in Score-P was straightforward. If tracing is enabled, any instrumented events are passed to the OTF-2 backend. Likewise, the Score-P parallel profiling implementation inherits from Scalasca. If profiling is enabled, the performance metric statistics are recorded for each call path. For the internal representation of the profile, Score-P uses the call-tree structure of Scalasca. Currently, Score-P records the inclusive metrics runtime and visits separately for each call path and thread. Additional statistics recorded per call path for each metric include the sum, the minimum, the maximum, the sum of square values, and the number of values. After the measurement run, the profile can be stored either in the CUBE-4 format or in the TAU snapshot format. When using the CUBE-4 format, the user can choose whether he wants to store only the sum (like in the former Scalasca measurement system) or a tuple of sum, min, max, and sum of squares which contains all data that is available from TAU profiles. The choice between profiling and tracing are specified via environment variables.

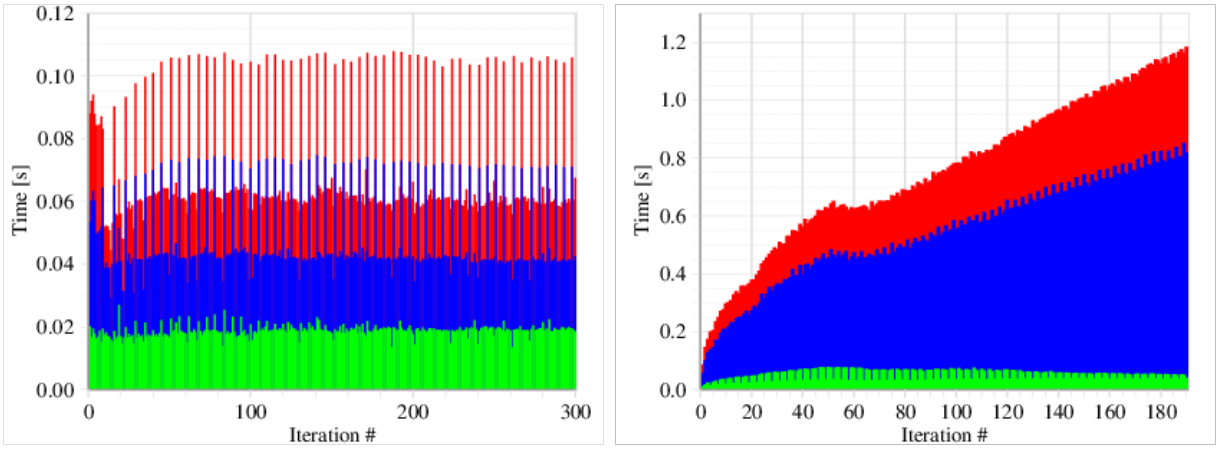
The profiling system supports the features:

- parameter-based profiling,
- dynamic regions, which makes every visit of a callpath create a different subtree,
- clustering of one dynamic region (see Section 5.2).
- phase profiling,
- recording of hardware/software counters and user defined metrics, and

- tasking (see Section 5.3).

To avoid expensive locking on every event which would inhibit scalability, the measurement system contains its own memory management system. Thus, every thread operates on a separate piece of memory. According to out tests (see Section 5.4), the Score-P measurement system is ready for peta-scale.

5.2 Time-Series Profiling



(a) 126.lammps (b) 129.tera_tf

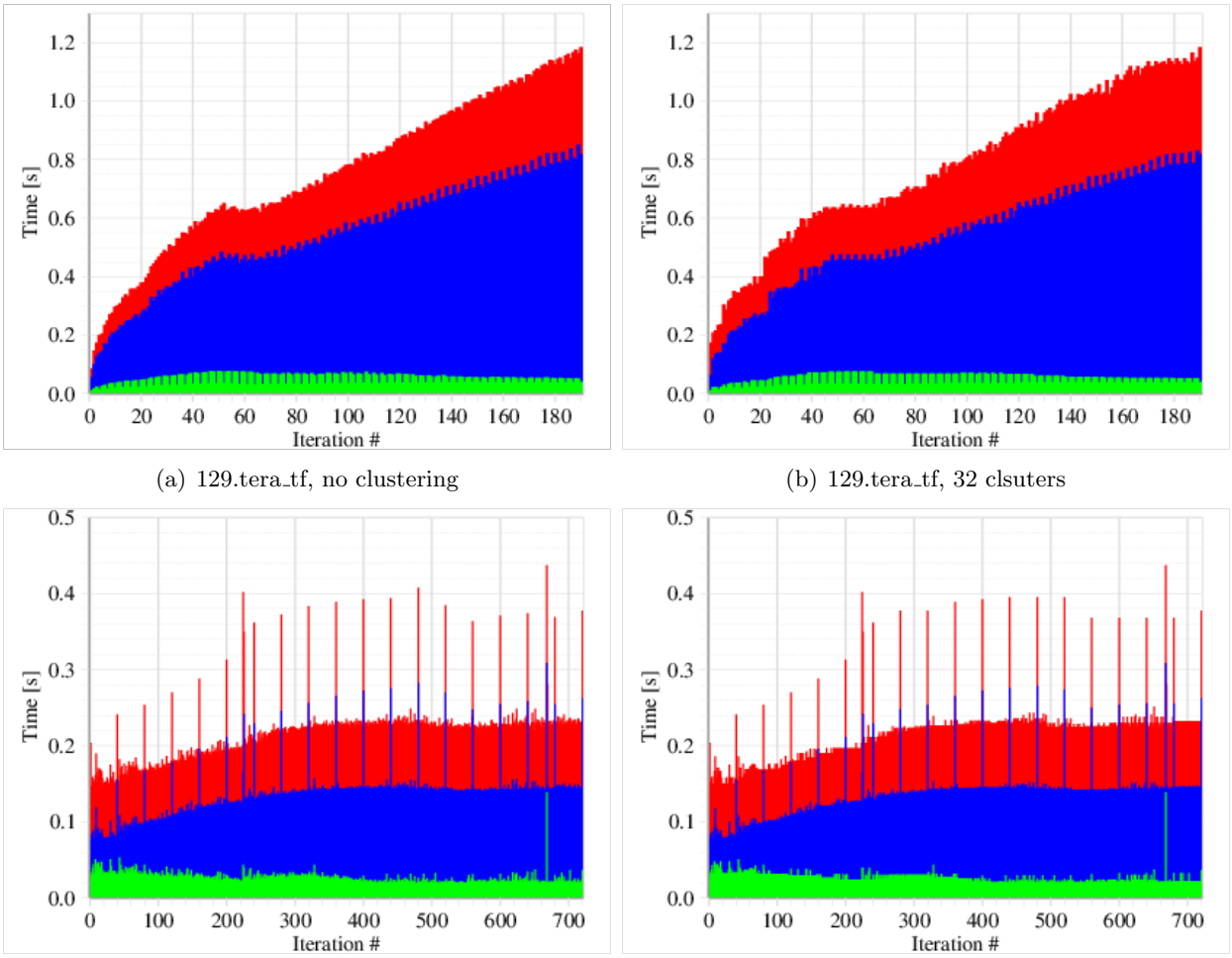
Figure 5: Runtime of iterations of the SPEC MPI benchmark codes 126.lammps and 129.tera_tf

The goal of time-series profiling is to investigate the variation of performance over time. As long as the application executes different code regions for every time period, the profile shows distinct statistics for each time period. However, many applications contain a main loop in which the application spends most of the time. Usually, profiling tool designs assume that the iterations of an iterative application behave basically the same, and thus, all visits are well represented by the resulting statistics. However, this assumption is not always true. Szebebyi et al. [26] have shown on examples from the SPEC MPI benchmarks (see Figure 5) and on the coulomb solver PEPC [27] that some applications change their behavior for different iterations.

In order to support the analysis of time-dependent behavior of iterative applications, the body of the main loop can be marked as a *dynamic* region. This causes the Score-P profiling system to record a separate profile for every iteration of the loop. However, for long running applications with a large number of iterations, this may lead to a high memory consumption and large results. This can make it difficult to apply this mechanism to large scale applications.

To reduce the memory requirements of the profile, we added a mechanism which clusters similar iterations to a single profile sub-tree instance [25]. The maximum number of clusters is configurable. However, we only cluster iterations with a similar structure. Iterations which contain different call paths are never clustered together. The time dynamics can be reconstructed from a mapping table, which stores the cluster associated with each iteration.

In case of the SPEC MPI benchmarks, the clustered profiles match the profiles without clustering very well, even if only 64 clusters are used. Figure 6 shows the comparison between the runtimes of iterations with clustering and without clustering. In the case of clustering the more complex PEPC coulomb solver, count based metrics show already a good match when using 64 clusters. However, time based metrics require more clusters for a good match [25].



(a) 129.tera_tf, no clustering

(b) 129.tera_tf, 32 clsuters

(c) 147.l2wrf2, no clustering

(d) 147.l2wrf2, 64 clusters

Figure 6: comparison between the runtimes of each iteration without clustering and with clustering for the SPEC MPI benchmarks 129.tera_tf, and 147.l2wrf2.

5.3 OpenMP task analysis

With the OpenMP specification 3.0 [6] the tasking construct was introduced. Using tasks, the programmer is able to express parallelism in his code at a much finer level of detail. Instead of specifying a single command stream per thread, as with the traditional parallel and work-sharing constructs, the programmer can now decompose his program into smaller tasks and specify dependencies between creator tasks and their children. The defined tasks are assigned to the available threads by the runtime system. This approach is supposed to automatically improve load balancing, although it incurs additional overhead in the runtime system. However, a tool must deal with the additional level of parallelism, otherwise it would disrupt any measurements taken. Furthermore, a new programming paradigm implies that new analysis techniques for that model are required because each programming model has its specific performance critical characteristics. We identified typical performance bottlenecks in OpenMP tasking applications in Section 5.3.1 and published this also in [22]. This is a prerequisite to any specialized targeted analysis feature.

As first step, we developed a portable method to distinguish individual task instances and to track their suspension and resumption using event-based instrumentation [15]. A prerequisite for

this approach is that tied tasks are used or untied tasks which are only suspended at task scheduling points. Based on this method, we developed a new event model for tasks. This event model was also published in [22]. We designed the event model that it can also be used for other task-based systems like OmpSs, or GPU kernels from CUDA. This event model was implemented in Score-P and OTF2.

The profiling system was extended to work with tasks.

5.3.1 Possible performance bottlenecks with OpenMP tasks

In task-parallel programs, typically many more task instances than compute resources exist. Consequently, we cannot expect all task instances to be executed in parallel. Tasks which have to wait at a synchronization point do therefore not necessarily indicate a performance drawback. In most HPC applications, the number of active threads is a good indication for the number of available compute resources, as most applications start one thread per core they want to use. Accordingly, all threads can be active at the same time. What needs to be investigated, even in tasking programs, is whether all threads are doing useful work all the time. Here, useful work means everything except spending time in the OpenMP runtime or doing nothing. The following three performance problems related to tasking can lead to situations where threads waste compute resources.

Too Finely Grained Task Parallelism. Overhead spent in the OpenMP runtime to create a task or to suspend and resume it should be avoided if possible. If the execution time of a task is very small, this overhead can consume more CPU cycles than the task's actual execution. In this case, it would be more efficient to execute the task's body immediately without separating it into a task. The overhead to create and manage a task, of course, depends on many different factors, such as the hardware, the compiler, the data-sharing attributes of the task, and so on. Thus, we cannot quantify precisely when it is beneficial to create a task.

Too Coarsely Grained Task Parallelism. In contrast to the previous situation, creating only a few very large tasks may result in load imbalance. For example, if 12 equally sized large tasks are created and eight threads are used, half of the threads will execute two tasks and the rest will only execute one. Even if there is a task for every thread, sometimes there might not be enough to smooth differences in the runtime of individual tasks, which can depend on dynamic conditions.

Task-Creation Bottleneck. When a lot of threads execute tasks while only a few threads create them, the creation of tasks can become the bottleneck. This can happen, for example, when tasks are created in a `single` region by just one thread. For `n` worker threads, the master thread must produce the tasks at least `(n-1)` times faster than they are executed by workers. This situation is commonly known in master-worker approaches where the master can become the performance bottleneck if the number of workers is too large. Another reason why not enough tasks are created might be a shortage of available parallelism in dynamic algorithms.

5.3.2 Profiling of Tasks

We provide the first profiling tool that provides call-path-level statistics about applications with OpenMP 3.0 tasks. The algorithm of the profiling algorithms are published in [16]. An example for the resulting task profile is shown in Figure 7.

The basic idea is to present every task construct as a separate tree. In the synchronization constructs in the implicit task, which is rooted in `main`, we add child nodes which indicate the

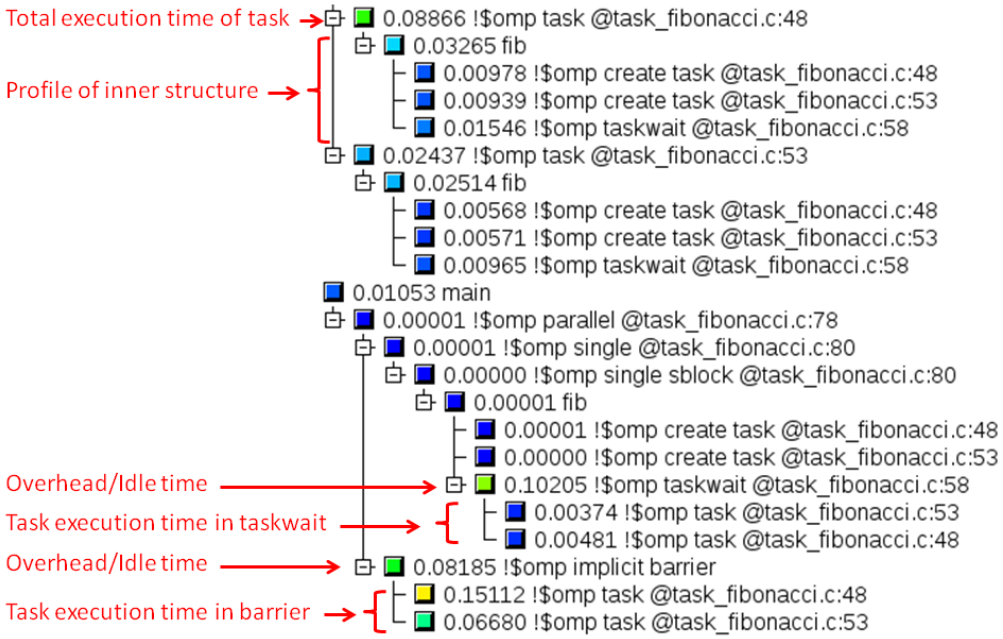


Figure 7: The profile of an example code using tasks.

time spend executing tasks inside this synchronization construct. Thus, it is possible to distinguish between time spend on executing user code and time spend in the runtime or idling. Furthermore, the inner structure of the tasks is shown, which allows traditional performance analysis within the tasks, like identifying hotspots.

5.4 Scaling of the Score-P measurement system

The Score-P infrastructure aims for peta-scale applications. For the instrumentation layer, it does not matter on how many cores the application runs. However, the measurement system has to run at the same scale as the application it measures.

In order to verify the scaling of the measurement system, we measured the initialization time, the run-time overhead during the application execution and the finalization time. Because the measurement system does not communicate among processes during initialization and application execution, the scale should not influence these parts. During finalization, the measurement system has to unify definitions and write all data to disk. Because the amount of data grows linearly with the number of processes, we expected a linear increase during finalization.

For testing we chose the sweep3d application and executed the measurements on the BlueGene/Q system Juqueen [2]. We performed measurements from 1024 processes up to 262144 processes. The measurements met our expectations: The initialization time of 1.9 ms was constant for all number of processes. The overhead remains constant around 5% and the finalization time grows linear. The overhead and the finalization time are shown in Figure 8 and Figure 9.

We made the tests in profiling mode and in tracing mode. For tracing the finalization time is much higher. However, this is to be expected, because the amount of data that is written to disk is much higher when tracing.

In addition, we verified the scaling of the Scalasca2 trace analyzer with the same setup on Juqueen, analyzing sweep3d runs from 1024 to 262144 processes. The results are shown in Figure 10.

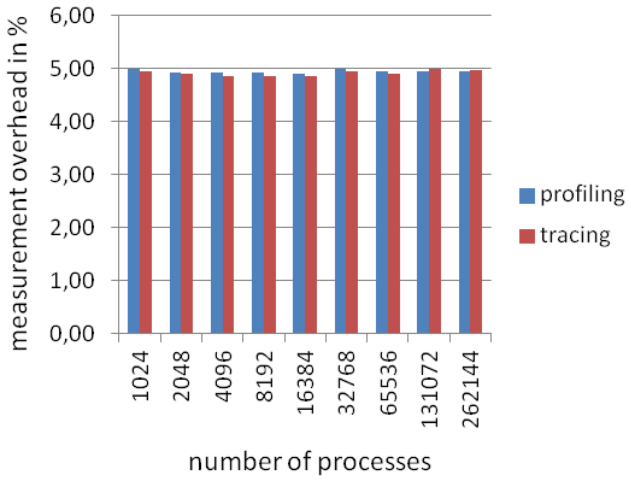


Figure 8: The measurement overhead of the Score-P measurement system depending on the number of processes on Juqueen when measuring sweep3d.

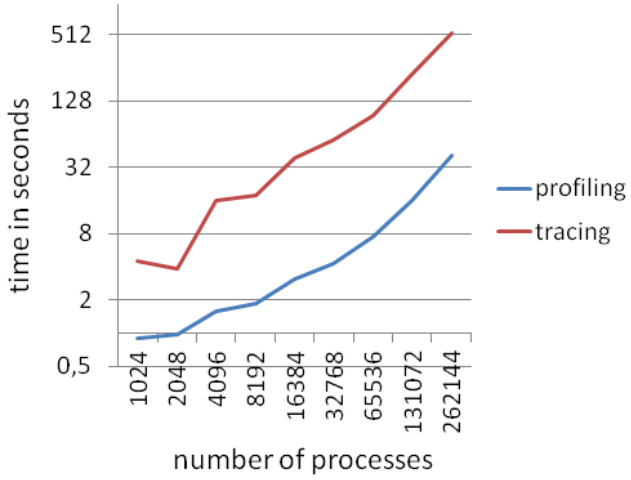


Figure 9: The finalization time of the Score-P measurement system depending on the number of processes on Juqueen.

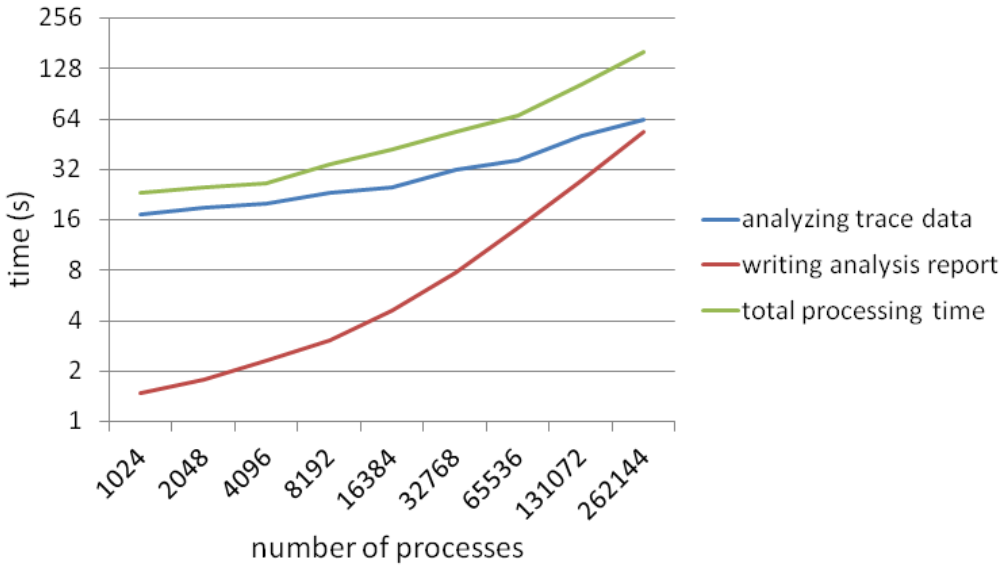


Figure 10: The time Scalasca2 needes to analyze traces of sweep3d run.

It shows the total time and a break down into analysis time and time spent for writing the profile data. It shows that Scalasca2 scales as well. However, we expect that we can improve the scalability even further, when the support of SIONlib, a parallel IO library, is fully integrated in Score-P and Scalasca.

6 aPRIMA

The refactoring and integration of the instrumentation and measurement components of TAU and Scalasca have been the primary work activities of the first two years of PRIMA. Our efforts on the analysis components is tied to the progress here because of the need to define common formats for the measurement data produced. Thus, the integration of the new formats into our analysis happened mainly in the last year. The following discusses the new formats we created for both profiles and traces that serve as the basis for analysis integration. Afterwards we report about the status of the integration of the new formats into our analysis tools. Finally, we devoted attention to analysis workflows and report our progress here.

6.1 Parallel Data Formats

The usage of common data formats is a significant step towards interoperability of the tools. Because the proprietary formats of the separate tools focus on their special features, new common formats were developed which should support the features of all tools and also include the lessons learned with respect to scalability and performance. This results in two new formats: CUBE-4 parallel profile format, and OTF-2 trace format. Although tracing was not a primary goal in the initial proposal, it is a fundamental part of the common infrastructure. Thus, we present it as part of the overall work, though it was not done with the manpower funded by PRIMA.

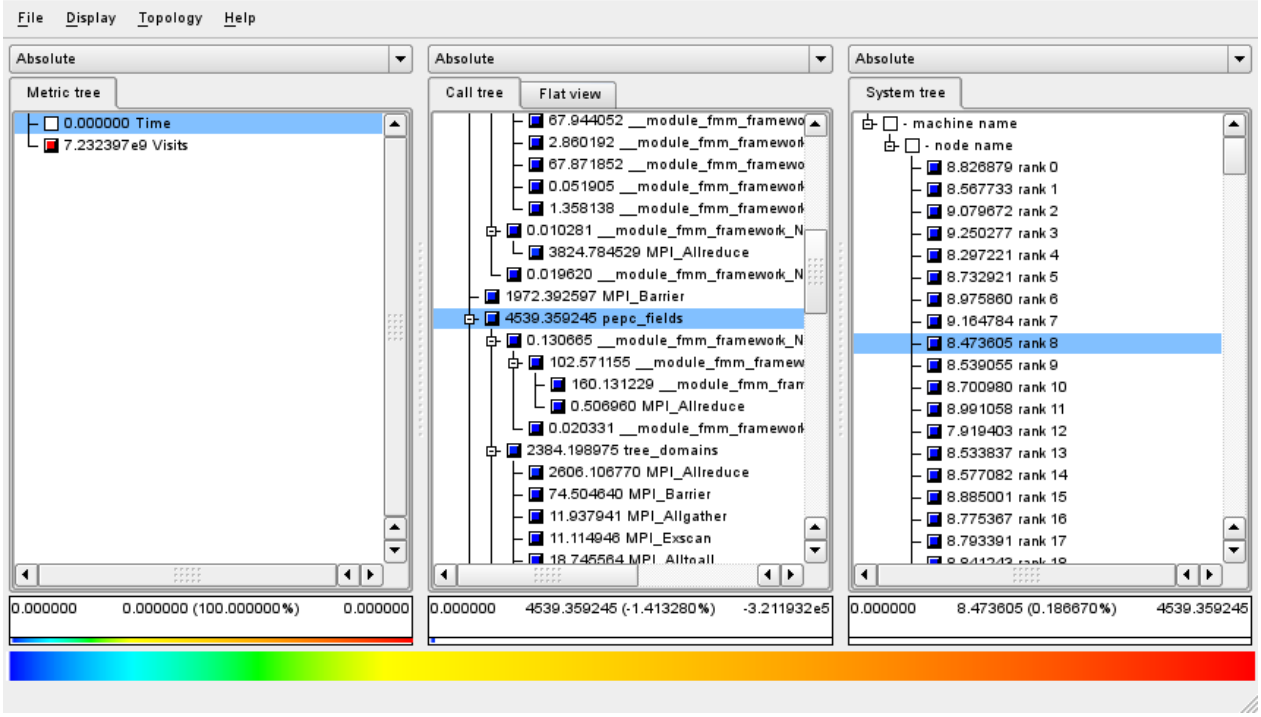


Figure 11: Snapshot of a CUBE-4 profile taken with Score-P on Jugene.

6.1.1 CUBE-4 Format

CUBE-4 is a highly scalable, memory efficient, flexible profile format. It is provided with reader and writer libraries, and a set of tools to compare, manipulate, or convert CUBE-4 files. Furthermore, the CUBE-4 package contains a graphical user interface (see Figure 11) to visualize the data. It is the new standard profile format for Score-P. CUBE-4 is distributed as a separate package to encourage other performance-tool developers to adopt the format for their own tools.

CUBE-4 evolved from the CUBE-3 profile format [24], the former Scalasca profile format. It preserves the basic CUBE-3 data model, but replaces many of the internal mechanisms to overcome limitations with respect to data size, performance and scalability. In the CUBE data model, for every metric the inclusive or exclusive value is stored per call path and executed thread. From this data, the exclusive or the inclusive metric values can be calculated, respectively. Each of the three dimensions (metric, call path, thread) has a hierarchical structure which is shown as a tree that can be interactively examined.

Most important technical changes that led to the improvements are:

- Clear separation of meta data and data definitions from the data itself
- Change from an XML-based to a binary representation of the data

Reader or writer now have random access to data items. This allows to load data only partially, overcoming the size limitation and increasing scalability. Furthermore, non-sequential reading of data becomes efficient because fast seeking methods are possible.

One of the milestones is to implement a scalable profile output algorithm. With the improved format and its implementation, writing large-scale runs became possible and the scalability and performance was improved. We also prototyped an algorithm that allows parallel writing of the profile

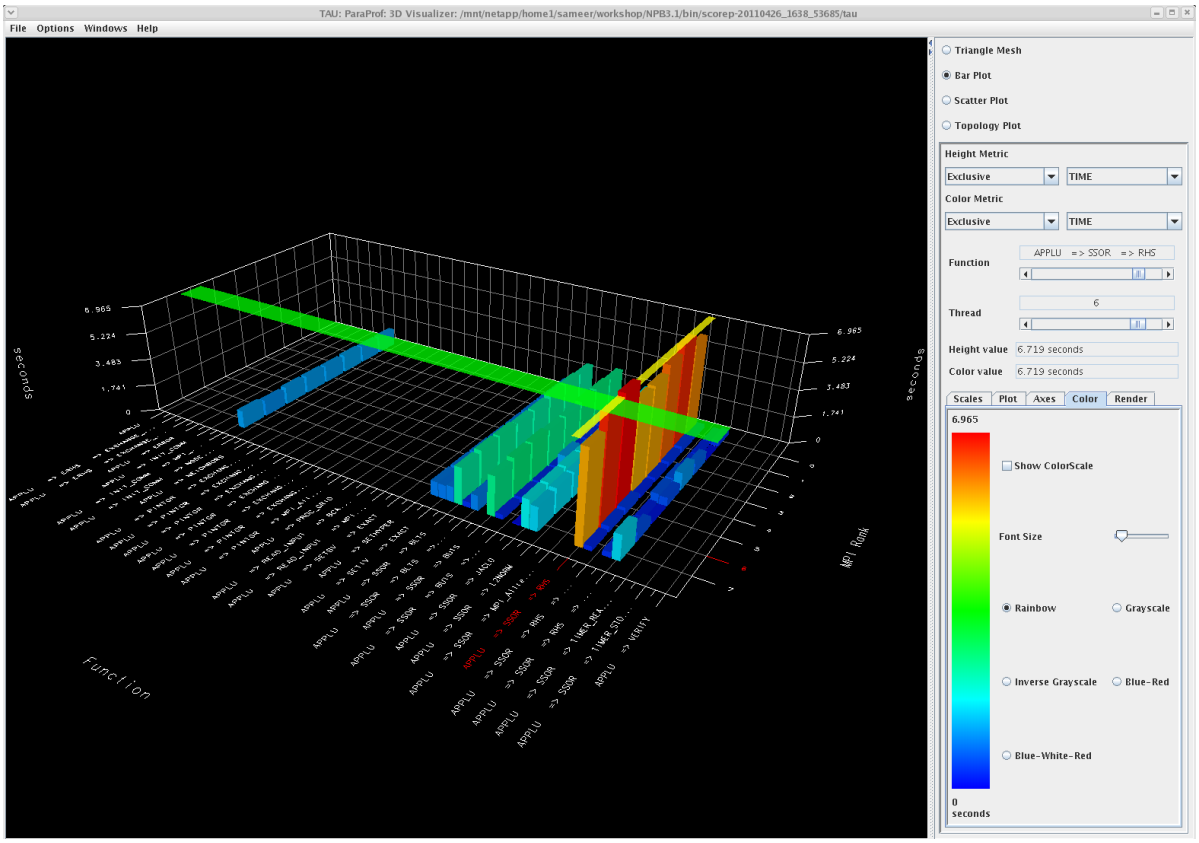


Figure 12: Sweep3D callpath profile generated using PDT's source code instrumentation

with multiple processes. However, the parallel writing did not yield any performance improvement due to file system synchronization.

The implementation of the internal changes are completed and first released in January 2012. The CUBE-4 writer is integrated into Score-P, which is able to write CUBE-4 profiles. Thus, the Score-P tests implicitly tested the CUBE-4 package and the results presented in Section 5.4 imply that CUBE-4 scales to the same level.

To support the full functionality of the TAU analysis tools, We extended CUBE-4 to support the full metric set of the TAU profiling format consisting of minimum, maximum and mean values and added the possibility to attach meta data for each thread to CUBE 4 profiles. Furthermore we added a Java reader library which is a prerequisite for the TAU analysis tool to read CUBE-4 profiles. The Score-P measurement system uses CUBE-4 as its primary profile output format. Additionally, we integrated it into Scalasca in the last year.

As a demonstration of the CUBE-4 format, callpath profiles generated by Score-P can be viewed by TAU's ParaProf tool, as shown in Figure 12 for the NAS Parallel Benchmark LU program. Here a 3D window shows the routines along one axis, the MPI ranks along the other and height and color represent performance metrics such as the exclusive time spent in a routine or a callpath.

6.1.2 OTF-2 Format

The basic idea for OTF-2 was to design a trace format which can serve as a common data source for the trace analysis tools Vampir and Scalasca. OTF-2 provides a library to read and write OTF-2

traces and handle the complex trace processing transparently, thus supporting the programmer with easy access to the trace data. Another major goal was to design the library in a much more flexible, efficient, and scalable way than OTF and EPILOG, which are the former proprietary formats of Vampir and Scalasca, respectively.

Behind a uniform interface, plug-in techniques for different backends are implemented which allow, for example, different compression methods or backward compatibility. In the last year we added backends for SIONlib [10], which is a scalable library for parallel reading and writing. Currently, the usage of SIONlib is limited to MPI only jobs, because of limited support of the hybrid mode in SIONlib itself.

Each module was tested separately and also partially peer-reviewed during the implementation phase. Because Score-P writes its traces in OTF-2 format the scalability and platform tests implicitly tested OTF-2. In the last year we integrated OTF-2 into Scalasca and Vampir. Since version 2.0, which was released August 2013, Scalasca uses Score-P as its instrumentation and measurement system. Vampir is based based on OTF-2 traces since version 8.0, too.

6.2 Analysis tools testing and release

In order to adopt the common infrastructure for our analysis tools, we implemented capabilities to read and write the new formats. For Scalasca it meant that we had to change the input format to OTF2 and the output format, which became CUBE4. Because the Score-P measurement system and the OTF2 format are more flexible than Scalasa's prior measurement system, we had to renew the internals of Scalasca, as well. As result, we refactored some parts of the Scalsaca tool and create a new major version of Scalasca, called Scalasca2.

We completed implementation of all features of Scalasca in the new Scalasca2. After releasing beta version in alread earlier, and used it already in tutorials, we finally released Scalasca 2.0 in August 2013. Meanwhile, Scalasca was successfully used in multiple trainigs where users brought their own codes. According to our tests, Scalasca2 scales well (see Section 5.4).

6.3 Integration of analysis tools and workflows

To address the PRIMA work item for analysis workflow, we decided to build on our work with the *Eclipse* integrated development environment (IDE). In recent years its functionality has expanded to support multiple languages and programming paradigms. The C/C++ Development Tools (CDT) and Photran projects provide functionality for C/C++ and Fortran development, respectively. The Parallel Tools Platform (PTP) project extends the capabilities of the CDT and Photran by offering parallel development, launch and debugging. These tools make Eclipse a viable means of enhancing traditional command line tools with IDE based development techniques in high performance computing.

We have developed an XML interface for defining both performance tool workflows and their UIs within Eclipse/PTP. The result is the general-purpose External Tools Framework (ETFw). ETFw allows both tool and application developers to integrate performance analysis systems into an Eclipse environment without the effort and expertise that are required to develop new Eclipse plug-ins. In fact, XML workflow definitions for external performance tools can be added or updated without restarting the Eclipse platform.

The ETFw provides an extensible, modular, general system for defining performance analysis worklows. Workflow systems have been shown to increase efficiency and ease of use for complex computational activities composed of discrete steps. With the growing complexity of and need for performance analysis in all venues of software development, the benefits of workow techniques are

```
<tool name="Cube4">
        <compile>
                <allcompilers command="scorep --instrument" group="scorep"/>
        </compile>
        <analyze foralllike="profile.cubex" depth="1">
                <utility type="view" command="cube4" group="cube4">
                        <argument value="%%FILE%%"/>
                </utility>
        </analyze>
</tool>
```

Figure 13: Example of ETFw Workflow XML for SCORE-P.

quite appropriate. The expertise required to perform a given performance analysis task or series of tasks can be encapsulated in a workow definition for easy distribution and deployment. So long as the necessary tools are available on the system, the user need only select the desired workflow and set any necessary starting parameters. The framework we have developed offers a modular, extensible solution to the problem of performance analysis in IDEs by encapsulating existing command line based tools. In addition to the basic requirement of offering performance analysis tool functionality, it allows tools to be linked together in a workflow of performance analysis steps.

The ETFw's XML workflow format consists of three fundamental elements, which define the compilation, execution, and analysis steps of the workflow. The order, number, and presence of these steps may vary depending on the intent of the workflow and the employed analysis tools. The compilation step assigns compiler commands to be used for the relevant programming languages. The execution step defines commands to be composed with the target executable, if any. This covers tools such as Valgrind that take the target application as an input argument. It also allows the definition of environment variables which will be applied to the execution environment. The analysis step defines a series of commands that may be run on any data generated during program execution.

A basic workflow for the Score-P performance analysis system was straightforward to implement using the ETFw. Depending on the type of data collection to be used the environment of the execution phase and the applications invoked in the analysis phase may differ. In the example illustrated here the workflow will build an instrumented binary, execute the binary in an environment which will result in CUBE-4 profile output and then load the profile output in the CUBE-4 analysis tool. We have defined similar workflows for creation and visualization of OTF-2 trace files and TAU profile snapshots.

The usage of Score-P with the ETFw is very similar to the procedure for executing a project in Eclipse. Starting in the Eclipse workspace select the profile button, analogous to the run and debug buttons. Create a new profile configuration and adjust the usual Eclipse execution parameters if necessary. The performance analysis tab provided by the ETFw select the appropraite tool workflow, in this case CUBE-4. The application will be recompiled using the Score-P application, executed with environment variables appropriate to genreate the desired profile output and then the CUBE-4 application will be run on the resulting data.

By providing a general framework for performance analysis in a popular IDE we hope to simplify the process of performance analysis just as IDEs have assisted with simplifying other aspects of the software development cycle. Ideally this work can benefit not only existing software developers, but will also be of use to newcomers to high performance software engineering who may be more accustomed to IDE-based software development. Perhaps more importantly, we have created a means for expert users to encapsulate their expertise in a programmatic way; the creation of custom

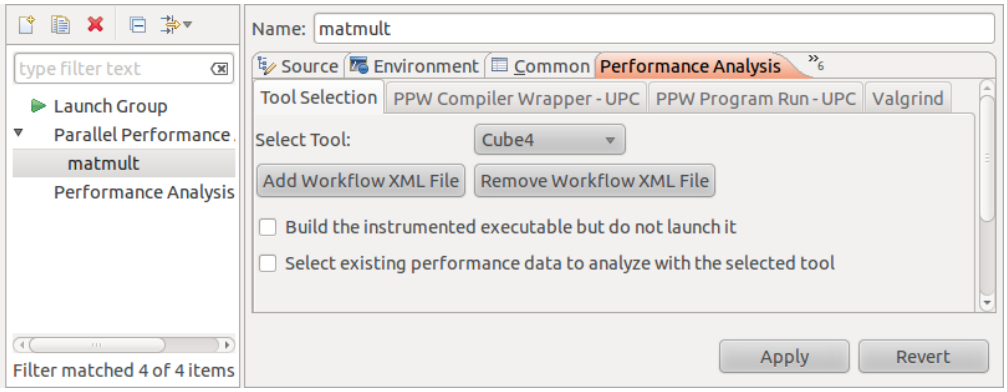


Figure 14: Tool workflow selection of ETFw launch configuration.

analysis workflows. A given series of performance analysis operations can be defined as a workflow by a tool developer or advanced user. This can then be made available to others who desire the final output of the potentially complex performance analysis procedure, but have no desire to engage in the manual, multi-step process every time they need to collect the data from their application.

7 Dissemination

7.1 Presentations

- Sameer Shende: Goals for PRIMA. At the PRIMA and SILC project meeting. 3rd February 2010, T.U. Munich, Garching, Germany.
- Jan Mußler: A binary instrumenter for Scalasca based on Dyninst. At the Paradyn/Condor week, 12th April 2010, Madison, Wisconsin, USA.
- Sameer Shende: TAU. At the 11th DOE ACTS Workshop: High Performance Software Tools to Fast-Track Development of Scalable and Sustainable Applciations, 20th August 2010, U.C. Berkeley, CA, USA.
- Sameer Shende: TAU: Performance evaluation at the extreme scale. Presentation at the Exascale: Runtime and Tools Requirements for the Programming Models of the Future Workshop at LACSS Conference, 13th October 2013, Santa Fe, NM, USA.
- Andreas Knüpfer and Felix Wolf: The future of the Open Trace Format (OTF) and open event trace recording. BoF at Supercomputing Conference, 16th November 2010.
- Sameer Shende: Scaling performance evaluation tool technology: A perspective from TAU. Workshop III: Reaching Exascale in this Decade, HiPC Conference, India, 19th December 2010.
- Sameer Shende: Simplifying the usage of performance evaluation tools: Experiences with TAU and DyninstAPI. At the Paradyn/Condor week, 14th April 2010, Madison, Wisconsin, USA.
- Daniel Lorenz: How to reconcile event-based performance analysis with tasking in OpenMP. At the 6th International Workshop on OpenMP (IWOMP), June 14-16 2010, Tsukuba, Japan.

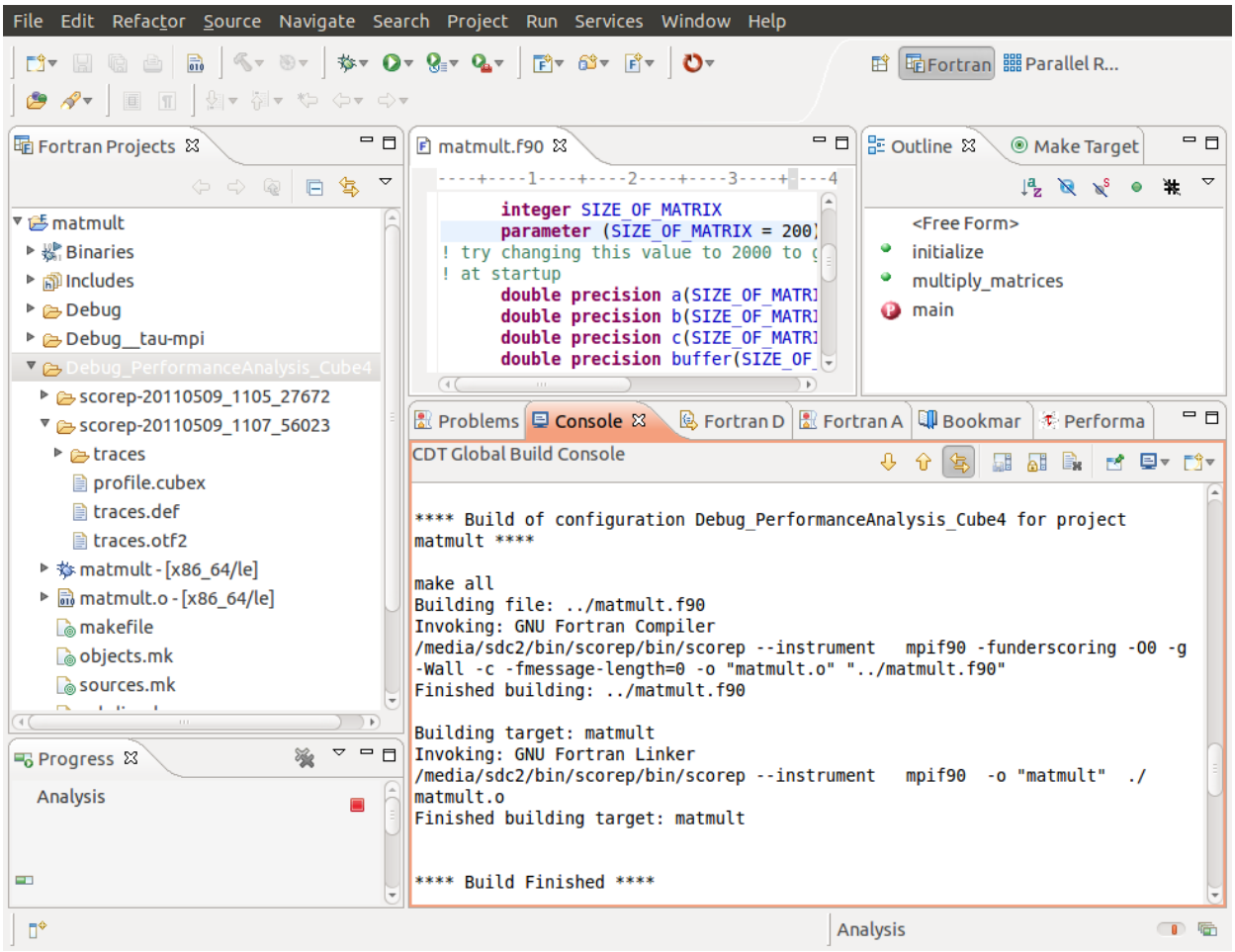


Figure 15: Eclipse workspace and build output from SCORE-P analysis run.

- Andreas Knüpfer: Score-P - A unified performance measurement system for petascale applications. Presentation at CiHPC: Competence in High Performance Computing, June 22-24 2010, Schwetzingen, Germany.
- Andreas Knüpfer: SILC: Skalierbare Infrastruktur zur automatischen Leistungsanalyse paralleler Codes: Status und Ausblick. At the BMBF status meeting, November 23-24 2010, Berlin, Germany.
- Sameer Shende: TAU performance system and PRIMA. At the PRIMA project Meeting, 2nd February 2011, Aachen, Germany.
- Daniel Lorenz: Reducing the overhead of direct application instrumentation using prior static analysis. At the Paradyn/Condor week, May 2-6 2011, Madison, Wisconsin, USA.
- Zoltán Szebenyi, Todd Gamblin, Martin Schulz, Bronis R. de Supinski, Felix Wolf, and Brian J. N. Wylie: Reconciling sampling and direct instrumentation for unintrusive call-path profiling of MPI programs. IPDPS 2011, Anchorage, AK, USA, May 18, 2011.
- Felix Wolf: Recent Scalasca research. DoE CScADS Workshop, Tahoe City, California, August 2, 2011.

- Sameer Shende: TAU performance system. At the Winter Workshop 2012 at the Argonne National Laboratory, ACLF, 26th January 2012, IL, USA.
- Sameer Shende: TAU. At the 13th DOE ACTS Workshop: Scalable and Robust Computational Libraries and Tools for High-End Computing, 14th August 2012, U.C. Berkeley, CA, USA.
- Daniel Lorenz: Reducing the overhead of direct application instrumentation using prior static analysis, Europar 2011, Bordeaux, France, August 31, 2011.
- Daniel Lorenz: Reducing the overhead of direct application instrumentation using prior static analysis, Exascale Computing Research Center at the University of Versailles, France, September 6, 2011.
- Felix Wolf: The role of performance engineering in HPC application development. Sino-German Workshop on Cloud-based High Performance Computing, Shanghai, China, September 28, 2011.
- Daniel Lorenz: Task analysis with Score-P. CScADS Tools Workshop, Snowbird, UT, USA, June 27, 2012.
- Felix Wolf: Scalable performance analysis of high-performance computing applications. RIKEN Advanced Institute for Computational Science, Kobe, Japan, July 23, 2012.
- Daniel Lorenz: Profiling of OpenMP tasks with Score-P. PSTI 2012, Pittsburgh, PA, USA, September 11, 2012.
- Bernd Mohr and Felix Wolf: From particle to continuum physics - Performance-analysis tools in the exascale age. Exascale Application and Software Conference (EASC), Edinburgh, Scotland, UK, April 10, 2013.
- Daniel Lorenz: Trace-based analysis of task dependency effects on performance. Petascale Tools Workshop, Madison, WI, USA, July 15, 2013.
- Alexandrou Calotoiu, Torsten Hoefler, Marius Poke, and Felix Wolf: Using automated performance modeling to find scalability bugs in complex codes. Petascale Tools Workshop, Madison, WI, USA, July 15, 2013.
- Felix Wolf: Two new performance re-engineering approaches around Scalasca. Cray Tech Forum, Cray Inc. St. Paul, MN, USA, July 18, 2013.
- Felix Wolf and Kai Diethelm: Performance dynamics of massively parallel codes. 3rd HPS status conference of the Gauss Alliance, Dresden, Germany, September 5, 2013

7.2 Publications

- Dieter an Mey, Scott Biersdorf, Christian Bischof, Kai Diethelm, Dominic Eschweiler, Michael Gerndt, Andreas Knüpfer, Daniel Lorenz, Allen D. Malony, Wolfgang E. Nagel, Yury Oleynik, Christian Rössel, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-P - A unified performance measurement system for petascale applications. In *Proc. of the CiHPC: Competence in High Performance Computing, HPC Status Konferenz der Gauß-Allianz e.V., Schwetzingen, Germany*, pages 1–12. Gauß-Allianz, Springer, June 2010.

- Sameer Shende, Allen D. Malony, and Alan Morris. Improving the scalability of performance evaluation tools. In *Proc. of the PARA 2010: State of the Art in Scientific and Parallel Computing, Part II: Minisymposium Scalable Tools for High Performance Computing*, Reykjavik, Iceland, volume 7134 of LNCS, Springer, pages 441–451, June 2010.

- Daniel Lorenz, Bernd Mohr, Christian Rössel, Dirk Schmidl, and Felix Wolf. How to reconcile event-based performance analysis with tasking in OpenMP. In *Proc. of 6th Int. Workshop of OpenMP (IWOMP), Tsukuba, Japan*, volume 6132 of *Lecture Notes in Computer Science*, pages 109–121. Springer, June 2010.

- Alan Morris, Allen D. Malony, Sameer Shende, and Kevin Huck. Design and implementation of a hybrid parallel performance measurement system. In *Proc. of the ICPP 2010 Conference*, pages 492–501, IEEE Computer Society, September 2010.

- Sameer Shende, Allen D. Malony, Alan Morris, Wyatt Spear, and Scott Biersdorff. TAU. In *Encyclopedia of Parallel Computing 2011*, pages 2025–2029. Springer, 2011.

- Allen D. Malony, Sameer Shende, Wyatt Spear, Chee Wai Lee, and Scott Biersdorff. Advances in the TAU performance system. In *Proc. of the Parallel Tools Workshop 2011*, pages 119–130. Springer, 2011.

- Sameer Shende, Allen D. Malony, Wyatt Spear, and Karen Schuchardt. Characterizing I/O performance using the TAU performance system. In *Proc. of the PARCO 2011 Workshop*, pages 647–655. IOS Press, 2011.

- Wyatt Spear, Allen D. Malony, Chee Wai Lee, Scott Biersdorff, and Sameer Shende. An approach to creating performance visualizations in a parallel profile analysis tool. In *Proc. of the Europar 2011 Workshop*, pages 156–165, LNCS, 2011.

- Jan Mußler, Daniel Lorenz, and Felix Wolf. Reducing the overhead of direct application instrumentation using prior static analysis. In *Proc. of the 17th Euro-Par Conference, Bordeaux, France*, volume 6852 of *Lecture Notes in Computer Science*, pages 65–76. Springer, September 2011.

- Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen D. Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer S. Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-P – A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Proc. of 5th Parallel Tools Workshop, 2011, Dresden, Germany*, pages 79–91. Springer Berlin Heidelberg, September 2012.

- Dirk Schmidl, Peter Philippen, Daniel Lorenz, Christian Rössel, Markus Geimer, Dieter an Mey, Bernd Mohr, and Felix Wolf. Performance analysis techniques for task-based OpenMP applications. In *8th Int. Workshop of OpenMP (IWOMP)*, volume 7312 of *LNCS*, pages 196–209, Berlin / Heidelberg, June 2012. Springer.

- Daniel Lorenz, Peter Philippen, Dirk Schmidl, and Felix Wolf. Profiling of OpenMP tasks with Score-P. In *Proc. of Third International Workshop of Parallel Software Tools and Tool Infrastructures (PSTI 2012), Pittsburgh, PA, USA*, September 2012.

- Daniel Lorenz, David Böhme, Bernd Mohr, Alexandre Strube, and Zoltán Szebenyi. Extending Scalasca's analysis features. In *Tools for High Performance Computing 2012*, pages 115-126, Springer Berlin Heidelberg, 2013.

- Alexandre Eichenberger, John Mellor-Crummey, Martin Schulz, Michael Wong, Nawal Copty, John DelSignore, Robert Dietrich, Xu Liu, Eugene Loh, and Daniel Lorenz. OMPT: OpenMP tools application programming interfaces for performance analysis. In *Proc. of the 9th International Workshop on OpenMP (IWOMP), Canberra, Australia*, pages 171–185, Berlin / Heidelberg, September 2013. Springer.

- N. A. Romero, C. Glinsvad, A. H. Larsen, J. Enkovaara, S. Shende, V. A. Morozogv, and J. J. Mortensen. Performance characterization of electronic structure calculations on massively parallel supercomputers: a case study of GPAW on the Blue Gene/P architecture. In *Concurrency and Computation: Practice and Experience*, John Wiley and Sons, December 2013.

- Andres Charif-Rubial, Denis Barthou, Cedric Valensi, Sameer Shende, Allen D. Malony, and William Jalby. MIL: A language to build program analysis tools through static binary instrumentation. In *Proc. HiPC 2013 Conference*, India, IEEE, December 2013.

Not directly funded through PRIMA, but still in the scope of its mission:

- Zoltán Szebenyi, Felix Wolf and Brian J. N. Wylie. Space-efficient time-series call-path profiling of parallel applications. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC09), Portland, OR, USA*. ACM, November 2009.

- Bernd Mohr, Brian J. N. Wylie, Felix Wolf. Performance measurement and analysis tools for extremely scalable systems. In *Concurrency and Computation: Practice and Experience*, 22(16):2212–2229, 2010, (ISC 2008 Award).

- Zoltán Szebenyi, Todd Gamblin, Martin Schulz, Bronis R. de Supinski, Felix Wolf and Brian J. N. Wylie. Reconciling sampling and direct instrumentation for unintrusive call-path profiling of MPI programs. *Proc. of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS), Anchorage, AK, USA*. IEEE, May 2011.

- Felix Wolf. Scalasca. In *Encyclopedia of Parallel Computing*, pages 1775–1785, Springer, October 2011.

- Allen D. Malony Metrics. In *Encyclopedia of Parallel Computing 2011*, pages 1124–1130. Springer, 2011.

- Jeff R. Hammond, Sriram Krishnamoorthy, Sameer Shende, Nichols A. Romero, and Allen D. Malony. Performance characterization of global address space applications: a case study with NWChem. In *Concurrency and Computation: Practice and Experience*, 24(2), pages 135–154, 2012.

- Markus Geimer, Pavel Saviankou, Alexandre Strube, Zoltán Szebenyi, Felix Wolf, Brian J. N. Wylie. Further improving the scalability of the Scalasca toolset. In *Proc. of PARA 2010: State of the Art in Scientific and Parallel Computing, Part II: Minisymposium Scalable tools for High Performance Computing*, Reykjavik, Iceland, June 2010, volume 7134 of Lecture Notes in Computer Science, pages 463–474, Springer, 2012.

- Daniel Lorenz, David Böhme, Bernd Mohr, Alexandre Strube, and Zoltán Szebenyi. Extending Scalascas analysis features. *Proc. of 6th Parallel Tools Workshop, 2012, Stuttgart, Germany.* pages 115–126. Springer Berlin Heidelberg, February 2013.

- Youssef Hatem. Critical path analysis of parallel applications using OpenMP tasks. Master thesis. German Research School for Simulation Science, Jülich Supercomputing Centre, and RWTH Aachen University. March 2013.

- Kevin Huck, Sameer Shende, Allen D. Malony, Hartmut Kaiser, Allan Porterfield, and Ron Brightwell. An early prototype of an autonomic performance environment for exascale. In *Proc. of the Runtime and Operating Systems for Supercomputers (ROSS'13) Workshop at ICS 2013*, Eugene, OR, ACM, USA, June 2013.

- Ahmad Qaqasmeh, Abid M. Malik, Barbara M. Chapman, Kevin A. Huck, and Allen D. Malony Open source task profiling by extending the OpenMP runtime API. In *Proc. of the IWOMP 2013*, pages 186-199, Springer, September 2013.

- David Ozog, Jeff R. Hammond, James Dinan, Pavan Balaji, Sameer Shende, and Allen D. Malony. Inspector-executor load balancing algorithms for block-sparse tensor contractions. In *Proc. of ICPP 2013 Conference*, Lyon, France, IEEE, pages 30–39, October 2013.

- Alexandru Calotoiu, Torsten Hoefler, Marius Poke, Felix Wolf. Using automated performance modeling to find scalability bugs in complex codes. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC13)*, Denver, CO, USA, ACM, November 2013.

- Andreas Knüpfer, Robert Dietrich, Jens Doleschal, Markus Geimer, Marc-André Hermanns, Christian Rössel, Ronny Tschüter, Bert Wesarg, Felix Wolf. Generic support for remote memory access operations in Score-P and OTF2. In *Tools for High Performance Computing 2012*, pages 57–74, Springer Berlin Heidelberg, 2013.

- Hari Radhakrishnan, Damian Rouson, Karla Morris, Sameer Shende, and S. Kassinos Test-driven coarray parallelization of a legacy Fortran application. In *Proc. of SE-HPCCSE 2013: Workshop on Software Engineering for Performance Computing in Computational Science andEngineering, SC 2013*, ACM SIGHPC, pages 33–40, November 2013.

- Sameer Shende and Allen D. Malony TAU. Chapter in *High Performance Parallel I/O*, eds. Prabhat, Quincey Koziol, (to appear), Chapman and Hall/CRC (June 2014).

7.3 Tutorials

- Sameer Shende, David Cronk, Rui Liu, and Nick Nystrom: Using POINT performance tools (PAPI, PerfSuite, TAU, Scalasca, and Vampir) to understand and optimize multicore codes. Full day tutorial at LCI 2010 Conference, 8th March 2010, Pittsburgh, PA, USA.

- I. Compres, M. Geimer, M. Gerndt, S. Shende, B. Wesarg: Hands-on practical hybrid parallel application performance engineering. EuroMPI, Vienna, Austria, September 23, 2012.

- D. Böhme, M. Geimer, M. Jurenz, Y. Oleynik, V. Petkov, P. Philippen, W. Spear, R. Tschüter, J. Weidendorfer, A. S. Charif-Rubial, E. Oseret: 10th VI-HPS Tuning Workshop, LRZ, Garching, October 16-19 2012.

- M. Geimer, P. Philippen, A. Knüpfer, T. William: Hands-on trace-based performance analysis with VAMPIR and Scalasca, SEA Conference, NCAR, Boulder, CO, USA, April 3-5, 2013.
- B. Wylie, A. Charif-Rubial, E. Oseret, R. Tschüter, M. Lücke, S. Shende, F. Winkler, V. Tsymbal: 11th VI-HPS Tuning Workshop, Maison de la Simulation, Saclay, France, April 22-25, 2013.
- B. Mohr, M. Schulz, B. Wylie: Supporting performance analysis & optimization on extreme-scale computer systems – current state & future directions, International Supercomputing Conference (ISC'13), Leipzig, Germany, June 16, 2013.
- P. Blood, C. Rössel: Hands-on performance analysis & optimization, 2013 International Summer School on HPC Challenges in Computational Sciences, New York University, NY, USA, June 23, 2013.
- B. Wylie, R. Dietrich: Parallel performance engineering, CSCS-USI Summer School, USI, Lugano, Switzerland, July 17-18, 2013.
- B. Wylie, S. Shende, D. Terpstra, F. Winkler: Hands-on practical hybrid parallel application performance engineering, XSEDE'13, San Diego, CA, USA, July 22, 2013.
- M. Geimer, M. Weber, F. Winkler: Program Analysis & Tuning Workshop, DKRZ, Hamburg, Germany, August 5-7, 2013.
- B. Wylie, M. Schulz, J. Protze, R. Tschüter, I. Comprés Ureña: Tools for high productivity supercomputing, Euro-Par'13, Aachen, Germany, August 26, 2013.
- B. Wylie, W. Frings, R. Tschüter, M. Geimer, S. Shende, D. Böhme, B. Wesarg, Y. Oleynik, J. Gimenez, H. Servat, A. Strube, J. Protze, F. Münchhalfen, A. Charif-Rubial, E. Oseret: 12th VI-HPS Tuning Workshop, JSC, Jülich, October 7-11 2013.

8 Unexpended Funds

8.1 University of Oregon

At the University of Oregon, we have no unexpended funds.

8.2 Forschungszentrum Jülich

At Forschungszentrum Jülich, we have no unexpended funds.

References

[1] INDEED web page. `http://gns-mbh.com/indeed.html`.

[2] Juqueen web page. `http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html`.

[3] PEPC web page. `http://wwwgsb.fz-juelich.de/ias/jsc/EN/AboutUs/Organisation/ComputationalScience/Simlabs/slpp/SoftwarePEPC/_node.html`.

[4] PFLOTRAN web page. `http://ees.lanl.gov/pflotran/`.

[5] Dieter an Mey, Scott Biersdorff, Christian Bischof, Kai Diethelm, Dominic Eschweiler, Michael Gerndt, Andreas Knüpfer, Daniel Lorenz, Allen D. Malony, Wolfgang E. Nagel, Yury Oleynik, Christian Rössel, Pavel Saviankou, Dirk Schmidl, Sameer S. Shende, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-P: A unified performance measurement system for petascale applications. In *Proc. of the CiHPC: Competence in High Performance Computing, HPC Status Konferenz der Gauß-Allianz e.V., Schwetzingen, Germany, June 2010*, pages 85–97. Gauß-Allianz, Springer, 2012.

[6] OpenMP Architecture Review Board. OpenMP application progam interface version 3.0. Technical report, OpenMP Architecture Review Board, May 2008.

[7] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14:189–204, 2000.

[8] T. Deselaers, D. Keysers, and H. Ney. Features for Image Retrieval - a quantitative comparison. In *26th DAGM Symposium, Pattern Recognition (DAGM 2004)*, number 3175 in Lecture Notes in Computer Science, pages 228 – 236, Tübingen, Germany, 2004.

[9] Alejandro Duran, Xavier Teruel, Roger Ferrer, Xavier Martorell, and Eduard Ayguadé. Barcelona OpenMP Tasks Suite: A Set of Benchmarks Targeting the Exploitation of Task Parallelism in OpenMP. In *38th International Conference on Parallel Processing (ICPP '09)*, pages 124–131, Vienna, Austria, September 2009. IEEE Computer Society, IEEE Computer Society.

[10] W. Frings, F. Wolf, and V. Petkov. Scalable massively parallel I/O to task-local files. In *Proc. ACM/IEEE SC'09 Conference (Portland, OR, USA)*, November 2009.

[11] Markus Geimer, Sameer S. Shende, Allen D. Malony, and Felix Wolf. A generic and configurable source-code instrumentation component. In *Proc. of the International Conference on Computational Science (ICCS)*, volume 5545 of *Lecture Notes in Computer Science*, pages 696–705. Springer, May 2009.

[12] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, April 2010.

[13] Michael Gerndt, Karl Fürlinger, and Edmond Kereku. Periscope: Advanced techniques for performance analysis. In *Parallel Computing: Current & Future Issues of High-End Computing, Proceedings of the International Conference ParCo 2005*, volume 33 of *NIC Series*, pages 15–26, October 2006.

[14] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen D. Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer S. Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-P – A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Proc. of 5th Parallel Tools Workshop, 2011, Dresden, Germany*, pages 79–91. Springer Berlin Heidelberg, September 2012.

[15] D. Lorenz, B. Mohr, C. Rössel, D. Schmidl, and F. Wolf. How to reconcile event-based performance analysis with tasking in OpenMP. In Mitsuhisa Sato, Toshihiro Hanawa, Matthias Müller, Barbara Chapman, and Bronis de Supinski, editors, *Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More*, volume 6132 of *Lecture Notes in Computer Science*, pages 109–121. Springer Berlin / Heidelberg, 2010.

[16] Daniel Lorenz, Peter Philippen, Dirk Schmidl, and Felix Wolf. Profiling of OpenMP tasks with Score-P. In *Proc. of Third International Workshop of Parallel Software Tools and Tool Infrastructures (PSTI 2012), Pittsburgh, PA, USA*, September 2012.

[17] MAQAO, 2012. `http://www.maqao.org/`.

[18] Bernd Mohr, Allen D. Malony, Sameer S. Shende, and Felix Wolf. Design and prototype of a performance tool interface for OpenMP. *The Journal of Supercomputing*, 23(1):105–128, August 2002.

[19] Jan Mußler, Daniel Lorenz, and Felix Wolf. Reducing the overhead of direct application instrumentation using prior static analysis. In *Proc. of the 17th Euro-Par Conference, Bordeaux, France*, volume 6852 of *Lecture Notes in Computer Science*, pages 65–76. Springer, September 2011.

[20] Wolfgang E. Nagel, Alfred Arnold, Michael Weber, Hans-Christian Hoppe, and Karl Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer 63*, 12(1):69–80, January 1996.

[21] MANTEVO Sandia National Laboratories, 2012. `http://www.mantevo.org/`.

[22] Dirk Schmidl, Peter Philippen, Daniel Lorenz, Christian Rössel, Markus Geimer, Dieter an Mey, Bernd Mohr, and Felix Wolf. Performance analysis techniques for task-based OpenMP applications. In *8th Int. Workshop of OpenMP (IWOMP)*, volume 7312 of *LNCS*, pages 196–209, Berlin / Heidelberg, June 2012. Springer.

[23] Sameer S. Shende and Allen D. Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–331, 2006.

[24] Fengguang Song, Felix Wolf, Nikhil Bhatia, Jack Dongarra, and Shirley Moore. An algebra for cross-experiment performance analysis. In *ICPP '04: Proceedings of the 2004 International Conference on Parallel Processing*, pages 63–72, Washington, DC, USA, 2004. IEEE Computer Society.

[25] Zoltán Szebenyi, Felix Wolf, and Brian J. N. Wylie. Space-efficient time-series call-path profiling of parallel applications. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC09), Portland, OR, USA*. ACM, November 2009.

[26] Zoltán Szebenyi, Brian J. N. Wylie, and Felix Wolf. SCALASCA parallel performance analyses of SPEC MPI2007 applications. In *Proc. of the 1st SPEC International Performance Evaluation Workshop (SIPEW), Darmstadt, Germany*, volume 5119 of *Lecture Notes in Computer Science*, pages 99–123. Springer, June 2008.

[27] Zoltán Szebenyi, Brian J. N. Wylie, and Felix Wolf. Scalasca parallel performance analyses of PEPC. In *Proc. of the 1st Workshop on Productivity and Performance (PROPER) in conjunction with Euro-Par 2008, Las Palmas de Gran Canaria, Spain*, volume 5415 of *Lecture Notes in Computer Science*, pages 305–314. Springer, 2009.

[28] VI-HPS. Virtual Institute - High Productivity Supercomputing. http://www.vi-hps.org, 2011.