D3.3.1 – Performance Analysis Tools design document

WP3: Development environment

Project Acronym	CRESTA
Project Title	Collaborative Research Into Exascale Systemware, Tools and Applications
Project Number	287703
Instrument	Collaborative project
Thematic Priority	ICT-2011.9.13 Exascale computing, software and simulation

Due date:	M10
Submission date:	31/07/2012
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organisation	TUD
Version:	1.0
Status	Final
Author(s):	Jens Doleschal (TUD)
Reviewer(s)	Joerg Hertzer (USTUTT), Adam Carter (UEDIN)

Dissemination level	
<PU/PP/RE/CO>	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	30/05/2012	First version of the deliverable	Jens Doleschal (TUD)
0.2	07/06/2012	New structure of the deliverable in accordance with the information /suggestions from WP3 telephone conference from June 5th.	Jens Doleschal (TUD)
0.3	18/06/2012	Created global fault-tolerance section since detailed information are not available yet.	Jens Doleschal (TUD)
0.4	30/06/2012	Redesign of Performance Counter section with the information provided by Cray.	Jens Doleschal (TUD)
0.5	04/07/2012	Finalising draft	Jens Doleschal (TUD)
0.6.1	10/07/2012	Internal Review	Joerg Hertzer (USTUTT)
0.6.2	12/07/2012	Internal Review	Adam Carter (UEDIN)
1.0	16/07/2012	Finalising deliverable considering review comments - Added more publications - Further explanations - Correction of typing errors - Reformatting references - Extension of the glossary	Jens Doleschal (TUD)

Table of Contents

Index of Figures

1 Introduction

In this document ("Performance Analysis Design Document, D.3.3.1") we present possible designs, planned modifications and extensions to the existing application performance analysis tools Score-P [1] and Vampir [2] to address scalability and heterogeneity.

We organised the document as follows: in Section 2 we describe the designs and extensions for the performance monitoring tool Score-P, i.e. collection of different kinds of performance counter and integration within the monitoring system, reduction of the amount of data to address scalability issues identified within the gap analysis (D3.1), and what extension will be done to address applications' demands on heterogeneity. In Section 3 we will specify the designs and extensions in terms of scalability and heterogeneity of the performance analysis and visualisation tool Vampir. Within Section 4 we present how to ensure that any extensions that we provide are well-tested and suitable for productive use. Finally, in Section 5 we address the state of fault-tolerance.

1.1 Purpose

This deliverable will specify the ideas and extensions to the application performance analysis tools Score-P and Vampir to address the following key components to meet exascale performance analysis requirements:

- Covering relevant performance data by including additional data sources from the hardware and runtime level (processor, bus, memory subsystem, network, faults, recovery);
- Pre-selection and automatic reduction of monitoring data to cope with the limitations of data storage, data visualisation, and last but not least human perception;
- Automatic data analysis and visualisation processes that guide the user through the optimisation process.

1.2 Glossary of Acronyms

CAF	Co-Array Fortran
CCG	Complete Call Graph
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CUPTI	CUDA Profiling Tools Interface
D	Deliverable
DMAPP	Distributed Shared Memory Application
GPI	Global Address Space Programming Interface
GPU	Graphics Processing Unit
IOFSL	I/O Forwarding Scalability Layer
MPI	Message Passing Interface
NIC	Network Interface Controller
OPARI	OpenMP Pragma and Region Instrumentor
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
OTF	Open Trace Format
PAPI	Performance API
PGAS	Partitioned Global Address Space
Pthreads	POSIX threads
TAU	Tuning and Analysis Utilities
TUD	Technische Universität Dresden
UPC	Unified Parallel C

2 Performance Monitoring

Driven by the experiences, basically various scalability limitations, gathered from our predecessor performance measurement systems, e.g. VampirTrace and Scalasca, and their corresponding file formats OTF and EPILOG, TUD and other partners have developed a newly designed joint performance measurement system called Score-P (see Figure 1) [1] and its underlying file format OTF2 [3] with focus on scalability (e.g., efficient handling of MPI communicators, and a tree-based reduction for trace unification), and interoperability with various performance analysis tools for application performance monitoring and analysis. Within this project we will use Score-P as the main application monitoring system and will adept and extend it to the needs of exascale application performance analysis identified within CRESTA.

Monitoring highly parallel applications with Score-P can be easily done by instrumenting the application with measurement probes and linking against several runtime libraries. Currently, Score-P provides the following instrumentation techniques:

- Compiler instrumentation,
- MPI library interposition,
- OpenMP source code instrumentation using OPARI2
- Source code instrumentation via the TAU instrumentor, and
- User instrumentation using convenient macros,

to collect performance relevant data on C/C++ and Fortran codes. In addition, it is possible to record several hardware counters by using the PAPI interface.

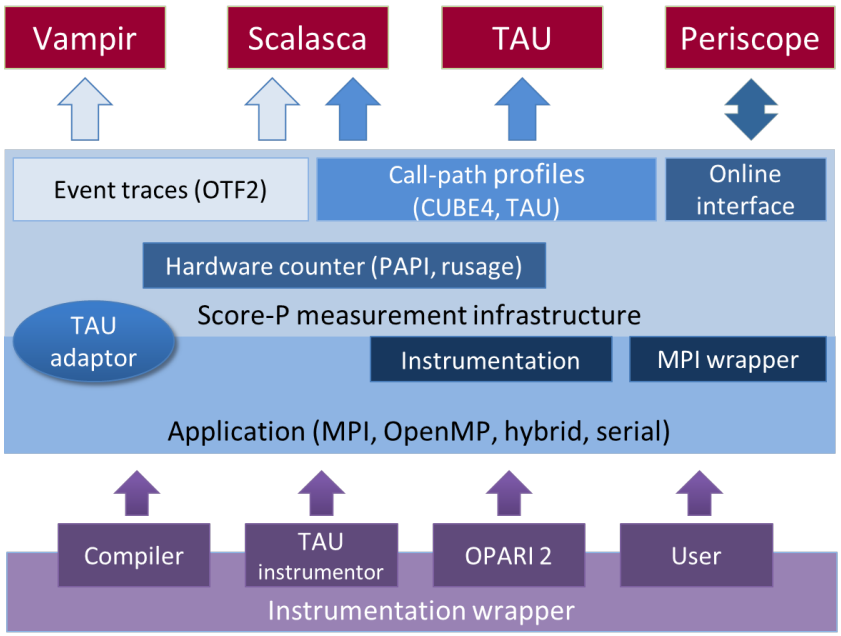

Figure 1 Architecture of the Score-P instrumentation and measurement system

2.1 Performance Data

Exascale systems will evolve toward complex, highly parallel and maybe also heterogeneous architectures that include multi-core or many-core nodes as well as heterogeneous accelerators such as GPUs. In such complex systems the use of hardware counters to gain insight into the highly parallel applications has become an accepted and integral method in the last decade [4].

PAPI (Performance API) and its components provide consistent platform and operating system-independent access to CPU hardware performance counters and hardware information of the machine subsystem, e.g. node-local I/O operations, that can be used in various performance monitoring tools. Traditionally, PAPI measures this information

on a per-process or thread basis even on context switches since most of the CPU hardware performance counters are core-local and exclusive.

Within today's more and more complex processor architectures the number of shared resources increases and there are also performance counters (e.g., uncore counter), which do not belong to a specific core. These counters cannot be measured via PAPI and also cannot be used as performance information (counter) for a specific process or thread. All performance information that does not belong to a specific core, like power of a node, temperature of a node, or on some architectures L3-Cache performance information have to be recorded separately and mapped to the corresponding process, and threads. To reduce the redundancy of performance information, to reduce the perturbation of hardware performance counter monitoring, and to increase the scalability of the performance monitoring it is advisable to record performance information of shared resources once and to map this information within the application monitoring to the corresponding locations.

Besides the hardware performance counters there exists a multitude of performance information within the machine sub-system. Cray provides for the Gemini network two categories of performance counters to the users [5]. Counters of the first category are NIC (Network Interface Controller) performance counters, which record information about the data moving through the network interface controllers. These counters reflect the network transfers beginning and ending on the node. The second category is network router tile counters. These atomic counters provide fine-granular information of each of the 48 tiles on a chip. Currently, these counters are only accessible via a proprietary interface and are at the moment from a user-view only available by using the CrayPat profiler [5][6].

Since there is no common interface for network performance counter like PAPI, we firstly plan to investigate which counters of the multitude of network counters are suitable for the application performance monitoring and how to measure them. In addition, it is advisable to map this information to the network topology of the machine. Therefore, we want to investigate techniques to detect the machine hardware and network topology and to include this information within the measurement system. This can be done either via a vendor topology API or a static machine description file.

In a second step, a proposal for a unique interface for network counters, maybe as a PAPI component or some other API that can be used within a wide range of monitoring tools either for global monitoring or application monitoring, would be very helpful. Because of the limited resources in this project this step will be postponed.

Depending on the latency and perturbation of performance counter requests and the locality of the performance information, the performance information can be measured during the application monitoring or can be added afterwards. Since it is at the moment not advisable to record Cray Gemini network counter within the application monitoring [5][7], and therefore to reduce the perturbation of the measurement, and network performance counter are basically not task-related, we want to record this information simultaneously with the application run and add it afterwards.

We are planning to extend the measurement system so that it is able to record hierarchical performance counters (if hardware topology information is available) to address the level of complexity of today's machine architectures and those of the future. We also plan to add an interface that allows us to add generic performance counters to an existing trace file.

2.2 Scalability

The continuation of the current fine-grained operation mode would produce an enormous amount of data that would overstress storage, performance analysis, and visualisation. As a result, for a plausible performance measurement only phases, events and processing elements of interest should be recorded.

In a first step, we want to identify hot spots, outliers, and irregularities to identify the regions of interest. In a second step, we want to use this information to steer the

instrumentation and measurement system for combined coarse-grained and fine-grained data collection, i.e. use selective instrumentation for the events of interest in combination with a dynamic filtering approach that can be used during runtime. The creation of automatic filtering rules is thereby mandatory.

Furthermore, we want to enable and disable the recording of the measurement to reduce the amount of data. We also want to be able to measure all phases of an application, e.g. all iterations of a loop, and only store phases of interest, which exceeds a predetermined threshold. Phases of non-interest will therefore be neglected.

In conclusion, we will research which steering and filtering decisions can be done at runtime or should be generated in a post-mortem step to steer the next performance measurement run.

Another research idea is to reduce the amount of data by combination of coarse-grained sampling and fine-grained event instrumentation for an optimal insight into the highly parallel applications. For the beginning we will focus on the previously mentioned selective instrumentation and dynamic runtime filtering with steering information from an irregularity analysis and will post-pone this topic.

2.3 Paradigms

To address the complexity and heterogeneity of exascale architectures new parallel programming paradigms arise or become more and more popular. To gain insight into these new dimensions of parallelism, performance measurement systems like Score-P have to be extended to measure the information of interest. Furthermore, they must be able to avoid additional communication and synchronisation overheads while setting up different namespaces to handle these different types of events without any loss of information. The information generation can be done either by instrumentation or sampling.

Since Score-P currently only provides an infrastructure for instrumentation techniques to intercept relevant events we will focus on instrumentation techniques for new parallel paradigms within the research, but nevertheless sampling is always an alternative.

2.3.1 Performance Monitoring of Pthreads

Monitoring of POSIX threads (Pthreads) can be easily done by using a classical function wrapping approach, where all POSIX calls will be renamed to the corresponding wrapping calls by the pre-processor and the wrapping calls are responsible to call the measurement system, which creates the corresponding events, and the original POSIX function.

Since Pthreads are using shared memory segments it is important that the monitoring system does not mix the events of concurrent threads and therefore must distinguish the threads from each other to ensure the consistency of the event buffers and chronology.

We are planning to extend the task identification model within Score-P so that it is able to handle multiple parallel paradigms used simultaneously within one application.

2.3.2 Performance Monitoring of Hardware Accelerators (CUDA/OpenCL)

In the last years CUDA/OpenCL capable devices have become more and more popular in the High Performance computing area. This is because they promise more floating point operations per seconds than a typical CPU will ever provide in a user application.

Host-side activities of OpenCL capable devices can be either monitored by instrumenting the library (if source code is available) or by using a shared library wrapper approach that uses the LD_PRELOAD mechanism.

Besides the host-based recording, some activities of the kernel can be monitored directly. For example, kernel execution and data transfers.

Monitoring of CUDA applications can be either done via the CUDA Profiling Tools Interface (CUPTI) [8] or by the previously mentioned library wrapping approach. CUPTI

provides different APIs that can be used to get insight into the CPU and GPU behaviour of CUDA applications. The benefits of CUPTI in comparison to the library wrapping approach are the reduced perturbation of the kernel execution and precise event (kernel) time information.

We plan to extend Score-P to monitor CUDA activities via CUPTI and OpenCL activities via a shared library wrapping approach. This requires the development of a generic task identification model, appropriate records, and a hardware accelerator measurement plug-in for Score-P.

2.3.3 Performance Monitoring of PGAS Languages

PGAS languages are available as library-based paradigms, e.g., GPI, and as language extensions, e.g., UPC, Co-Array Fortran.

Instrumentation of these language extensions can be done on different levels depending on how these extensions are implemented on a specific computing platform.

1. On systems where the language constructs are translated into calls to a runtime library, it is possible to intercept these library calls and record them. This can be done either by instrumenting the library or using a shared library wrapper approach that uses the LD_PRELOAD mechanism.
2. On systems where the language constructs are processed in the compiler runtime like on the Cray systems a library wrapping approach as proposed before will not work. In these cases it is possible to extend a source-to-source instrumentor like OPARI to instrument these language extensions [9].
3. Depending on the system implementation of these language extensions it is sometimes easier to wrap the underlying communication libraries, e.g. the DMAPP library on the Cray systems [10], and not to focus directly on these language constructs. The advantage is that this approach focuses on the network communication but in contrast some detailed information about the language constructs cannot be captured, e.g. one-sided in-memory communication of different processes in CAF running on the same node.

Since language extensions like UPC and CAF use very fine-granular operations, and to reduce perturbation within the measurement, we will focus on the monitoring of the PGAS network communication.

At the moment there is no primary candidate for PGAS within CRESTA and for the beginning we will focus on a shared library wrapping approach for the DMAPP library on Cray systems, which is used within many paradigms as a communication library, to gain indirect insight into the PGAS paradigms. Therefore, we have to investigate and develop one-sided communication models and their corresponding records that can be used for all parallel paradigms, which use remote memory access strategies.

2.3.4 Performance Monitoring of third-party libraries

The function call behaviour of third-party libraries, i.e. proprietary vendor libraries, where the calls are located in a runtime library, can be recorded by using a shared library wrapper that uses the LD_PRELOAD mechanism. We plan to extend Score-P to support library wrapping of shared libraries.

2.3.5 Extensions

1. Extension of Score-P to support library wrapping of shared libraries.
2. Research of one-sided communication model and their corresponding records.
3. Extension of Score-P to support hardware accelerators, Pthreads, and maybe further parallel programming models.
4. Generic task identification model that supports multiple parallel paradigms together.
5. Extension of Score-P by a counter interface that allows to add counters from global (shared) resources and also to add counters to existing traces in a scalable way.

2.4 Fault-tolerance

See Section 5 for more general information on fault-tolerance and fault-recovery.

3 Performance Analysis and visualisation

Investigating performance information of millions of concurrent processing elements will only be applicable for users by using intelligent data mining, data compression and visualisation strategies. Application performance visualizers like Vampir [2] come with various displays to gain insight into the highly complex behaviour of parallel applications in an intuitive way but are struggling with trillions of performance information. To handle the amounts of data gathered from the application monitoring, scalable and intelligent data analysis and processing techniques have to be developed.

3.1 Scalability

Offline performance analysis techniques have to handle amounts of information of a whole measurement run and usually store this information in its entirety on the parallel file system. Creating one file per measured location (e.g., process or thread) will no longer work on future file systems on a POSIX basis, because of the complex relationships of billions of entities with metadata and object data information organized on a folder structure. Therefore, we are evaluating different strategies like the use of the Sionlib [11] or IOFSL [12] to overcome scalability issues with future parallel file systems.

Bandwidth and capacity of the entire memory hierarchy – including the file system – are also limiting factors for offline performance analysis, as long as no information reduction is done. Data compression reduction techniques, such as the data reduction on the Complete Call Graph (CCG) data structure [13] will become a necessity at exascale in order to perform on-the-fly information reduction. We will develop a library that supports the creation of CCGs and investigate different visualisation alternatives for CCGs within Vampir.

The information of an entire time interval of millions of concurrent processing elements needs to be transferred and processed. Exascale performance analysis will require solutions to reduce the amount of data that tools display. One such option is to only display “interesting” locations that behave differently, which requires an automatic approach for pre-processing. Therefore, we want to investigate techniques for partial event trace visualisation. Also, visualisation of millions of different processing elements on a screen with a limited resolution is challenging and we want to research alternative visualisation strategies for highly parallel applications.

3.2 Paradigms

Each of the upcoming parallel paradigms typically comes with its own communication and synchronisation methods, enter and exit events. In addition, each parallel paradigm has its own specific performance behaviour where different kinds of performance issues may occur.

Investigation of all communication and synchronisation events within one view can result in a misleading view when one paradigm hides the performance behaviour of another paradigm. Therefore, we want to investigate possibilities in how to filter, summarise, and visualise performance information of heterogeneous applications, which use different paradigms concurrently, within Vampir (if Vampir is able to distinguish between the different event classes derived from the paradigms).

3.3 Fault-tolerance

In the case of hardware failures performance information will be corrupted and incomplete. Therefore, Vampir has to deal with this data in appropriate way, e.g. even on corrupted data functionalities like the message matching should work. Therefore, we want to investigate techniques that allows message matching on incomplete data, i.e. missing communication events. This can be done either by using piggy-backing techniques, i.e., extension of the original message by adding additional information to the message, additional messages, or by additional information provided by the MPI-3 standard.

For further information on fault-tolerance and fault-recovery see section 5.

3.4 Interaction and Interfaces

Profiling and statistical information can provide in a first step insight into the application. We intend to highlight these profiling and statistical results and locate outliers, irregularities and hot spots within Vampir.

4 Testing plan

For performance and scalability purposes we will use the benchmark suite from WP 2 D2.6.1 to test the performance monitoring and analysis tools.

Correctness and functionality tests are already implemented in the development flow of each tool and will be used continuously to ensure the needed quality for each CRESTA development branch (for each feature a separate branch).

4.1 Score-P

Score-P has its own quality management system and a continuous integration checker called bitten, a plugin of trac, internal review processes for new extensions and modifications, and also nightly builds and tests on various platforms provided by the Score-P partners.

4.2 Vampir and VampirServer

Vampir as a commercial product comes with its own quality management workflow and nightly builds on different platforms.

5 Fault-tolerance

Expectations for Exascale systems indicate that mean-time-to-failure may be far lower than a day. As a result, applications, systemware, operating system, and any type of runtime tool need to be aware of possible failures and may also need to recover from them. This may include the use of spare nodes or cores to replace failed components.

These effects need to be considered for the development of performance monitoring and analysis tools for exascale systems. Work package 2 evaluates operating system and programming model changes to handle such hardware faults in D2.5.2, which will be released in month 30 of the project. Thus, at the current project state, there is close to no indication how these mechanism and designs might look like. This includes an indication of a mean-time-to-failure that is to be expected. Also possible advances in hardware may render these concerns unnecessary altogether. As an immediate consequence we will not address any modifications for fault tolerance in this design document. Our hope is to use the insights that are available around project month 20, to include them in the second version of this design document (month 24).

6 References

[1] Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A.D., Nagel, W.E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S.S., Tschüter, R., Wagner, M., Wesarg, B., Wolf, F.: Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Proc. of 5th Parallel Tools Workshop*, 2011.

[2] Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H. Müller, M.S., Nagel, W.E: The Vampir Performance Analysis Tool-Set, Tools for High Performance Computing, Part 4, pp. 139-155, Springer Berlin/Heidelberg, 2008.

[3] Eschweiler, D., Wagner, M., Geimer, M., Knüpfer, A., Nagel, W.E.: Open Trace Format 2: The Next Generation of Scalable Trace Formats and Support Libraries, Vol. 22, Advances in Parallel Computing, pp. 481 - 490, ISBN: 978-1-61499-040-6, DOI: 10.3233/978-1-61499-041-3-481, 201.

[4] Terpstra, D., Jagode, H., You, H., Dongarra, J.: Collecting Performance Data with PAPI-C, Tools for High Performance Computing 2009, Springer Berlin / Heidelberg, 3rd Parallel Tools Workshop, Dresden, Germany, pp. 157-173, 2009.

[5] Overview of Gemini Hardware Counters, http://docs.cray.com/books/S-0025-10/ (June 2012).

[6] Using Cray Performance Analysis Tools, http://docs.cray.com/books/S-2376-41/S-2376-41.pdf (July 2012).

[7] Alverson, R., Roweth, D., Kaplan, L., The Gemini System Interconnect, *High* Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on High Performance Interconnects, pp.83-87, 18-20 Aug. 2010.

[8] CUPTI User's Guide, http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUPTI_Users_Guide.pdf, (June 2012).

[9] Mohr, B., DeRose, L., Vetter, J., A Performance Measurement Infrastructure for Co-Array Fortran, Proceedings of the International Conference on Parallel and Distributed Computing (Euro-Par 2005), Lisboa, Portugal, 30.08. - 02.09.2005. - Berlin, Springer, 2005, pp. 146 – 155.

[10] Using the GNI and DMAPP APIs, http://docs.cray.com/cgi-bin/craydoc.cgi?mode=Show;q=dmapp;f=books/S-2446-4003/html-S-2446-4003/S-2446-4003-toc.html#toc, (June 2012).

[11] Frings, W.; Wolf, F.; Petkov, V., Scalable Massively Parallel I/O to Task-Local File, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland, Oregon, November 14 - 20, 2009, SC'09, SESSION: Technical papers, Article No. 17, New York, ACM, 2009.

[12] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan, *Scalable I/O Forwarding* Framework for High-Performance Computing Systems, IEEE International Conference on Cluster Computing (Cluster 2009), New Orleans, LA, September 2009.

[13] Knüpfer, A., Nagel, W.E., Compressible memory data structures for event-based trace analysis, Future Generation Computer Systems (2006), ISSN: 0167-739X.