D3.3.2 – Performance Analysis Tools Design Document

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:	M24
Submission date:	30/09/2013
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	TUD
Version:	1.0
Status	Final
Author(s):	Jens Doleschal (TUD), Ronny Tschueter (TUD), Thomas Ilsche (TUD), Mario Bielert (TUD), Frank Winkler (TUD), Bert Wesarg (TUD), Holger Brunst (TUD), Harvey Richardson (CRAY), Berk Hess (KTH), George Mozdsynski (ECMWF), Xavi Aguilar (KTH), Adam Peplinski (KTH), Jiri Jaros (KTH), Michele Weiland (UEDIN), Lawrence Mitchell (UEDIN), Stephen Sachs (CRAY)
Reviewer(s)	Jason Beech-Brandt (CRAY), Peter Coveney (UCL)

Dissemination level	
PU	*PU – Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	16/07/12	Finale version of D3.3.1	Jens Doleschal (TUD)
0.2	23/08/13	Version of D3.3.2 for internal review	Jens Doleschal (TUD)
0.3	11/09/13	Addressed comments from internal review	Jason-Beech-Brandt (CRAY)
0.4	12/09/13	Addressed comments from internal review	Peter Coveney (UCL)
1.0	13/09/13	Final version for submission	Jens Doleschal (TUD)

Table of Contents

1 INTRODUCTION 5

1.1 PURPOSE 5
1.2 CHANGES TO D3.3.1 5
1.3 GLOSSARY OF ACRONYMS 6

2 PERFORMANCE MONITORING 7

2.1 PERFORMANCE DATA AND TOPOLOGY INFORMATION 7
2.2 SCALABILITY 9
2.3 PARALLEL PROGRAMMING MODELS 10
2.3.1 Performance Monitoring of Pthreads 10
2.3.2 Performance Monitoring of Hardware Accelerators (CUDA/OpenCL) 11
2.3.3 Performance Monitoring of PGAS Models 11
2.3.4 Performance Monitoring of third-party Libraries 14
2.3.5 Possible Future Extensions 14
2.4 FAULT-TOLERANCE 14

3 PERFORMANCE ANALYSIS AND VISUALISATION 15

3.1 SCALABILITY 15
3.2 PARALLEL PROGRAMMING MODELS 16
3.3 FAULT-TOLERANCE 17
3.4 INTERACTION AND INTERFACES 18

4 TESTING PLAN 19

4.1 SCORE-P 19
4.2 VAMPIR AND VAMPIRSERVER 19
4.3 PERFORMANCE MONITORING TESTS AND PERFORMANCE VISUALISATION RESULTS OF CRESTA APPLICATIONS 19
4.3.1 Performance Monitoring Test and Visualisation Result of Gromacs 19
4.3.2 Performance Monitoring Test and Visualisation Result of IFS 21
4.3.3 Performance Monitoring Test and Visualisation Result of Nek5000 22
4.3.4 Performance Monitoring Test and Visualisation Result of HemeLB 22
4.3.5 Performance Monitoring Test and Visualisation Result of OpenFoam 23

5 FAULT-TOLERANCE 24

6 REFERENCES 25

Index of Figures

Figure 1 Architecture of the Score-P instrumentation and measurement system. 7

Figure 2 Visualisation of the Cray hardware topology within Vampir's filter dialog. 9

Figure 3 Performance visualisation with Vampir of the application behaviour and node power and freshness information of a MPI parallel application using hyper-threading, running on four nodes with 32 processes each monitored on a Cray XC30. 9

Figure 4 Performance visualisation of the Cray DMAPP communication library with Vampir. It is important to see that dmapp_c_pset_test is called very often and therefore should not be recorded in detail to reduce the overhead of the monitoring. 12

Figure 5 Performance visualisation of a massive parallel bucket sort parallelized with Cray SHMEM. The master timeline shows very impressively the master-slave communication implemented with shmem_get64 operations coloured in light blue between the different processes surrounded by tow shmem_barrier_all operations coloured in yellow. 14

Figure 6 Vampir's Partial Loading Dialog showing the loading of the NEK5000 trace in the time range of 0 seconds to 1.6 seconds for the processes 512-1024. 16

Figure 7 Vampir's filtering dialog to filter threads of execution based on the paradigm. 17

Figure 8 Vampir's filtering dialog to filter threads of execution by their topology information. 17

Figure 9 Performance visualisation of a hybrid version of Gromacs parallelized with MPI, OpenMP, and CUDA running on 8 nodes with 16 cores and 2 Nvidia K20 each. Every process uses one Nvidia graphic card and sends its kernels coloured in blue to two different streams. 20

Figure 10 Performance visualisation with Vampir of the runtime behaviour of a hybrid Gromacs parallelized with MPI and OpenMP. One quarter of the processes and corresponding threads are responsible to compute the bonded interactions and integration (PME). 20

Figure 11 Vampir's filter dialog to select only the PME processes by using a regular expression. The resulting visualisation can be seen in Figure 12. 21

Figure 12 Performance visualisation of a selection of processes and threads (PME) with Vampir for a more detailed and focussed performance analysis without any "colouring noise" of the rest of the processes and threads. 21

Figure 13 Performance visualisation with Vampir of a hybrid IFS T1023 run. 22

Figure 14 Performance visualisation with Vampir of Nek5000 parallelised with MPI. .. 22

Figure 15 Performance visualisation with Vampir of HemeLB parallelised with MPI. .. 23

Figure 16 Performance visualisation with Vampir of OpenFoam parallelised with MPI. 23

1 Introduction

This document ("Performance Analysis Design Document, D.3.3.2") is an update of the previous D3.3.1. to 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 organized 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 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 Changes to D3.3.1

Section 2

- Changed architecture figure of Score-P to current state and added information about CUDA instrumentation

Section 2.1

- Added part for system topology on Cray systems
- Added part for metric plugin interface and example for power monitoring on Cray XC30

Section 2.2

- Added information for fault-tolerant matching scheme

Section 2.3.2

- Added information for monitoring CUDA activities via CUPTI and generic one-sided RMA event model

Section 2.3.3

- Extended PGAS section by addressing GASPI, SHMEM, and Chapel
- Introduced Score-P's generic one-sided RMA event model
- Added results from Co-Array-Fortran co-design team
- Added information for UPC monitoring
- Added sections 2.3.3.1 – 2.3.3.3

Section 3.1

- Added information for partial event analysis of large trace file data with Vampir and example with NEK5000 trace file

Section 3.2

- Added selective trace analysis by paradigm, location, name with Vampir and usage example with Gromacs

Section 4

- Added section 4.3

1.3 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
GASPI	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
RMA	Remote Memory Access
TAU	Tuning and Analysis Utilities
TUD	Technische Universitaet 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 **Error! Reference source not found.**) [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,
- CUDA instrumentation,
- 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.

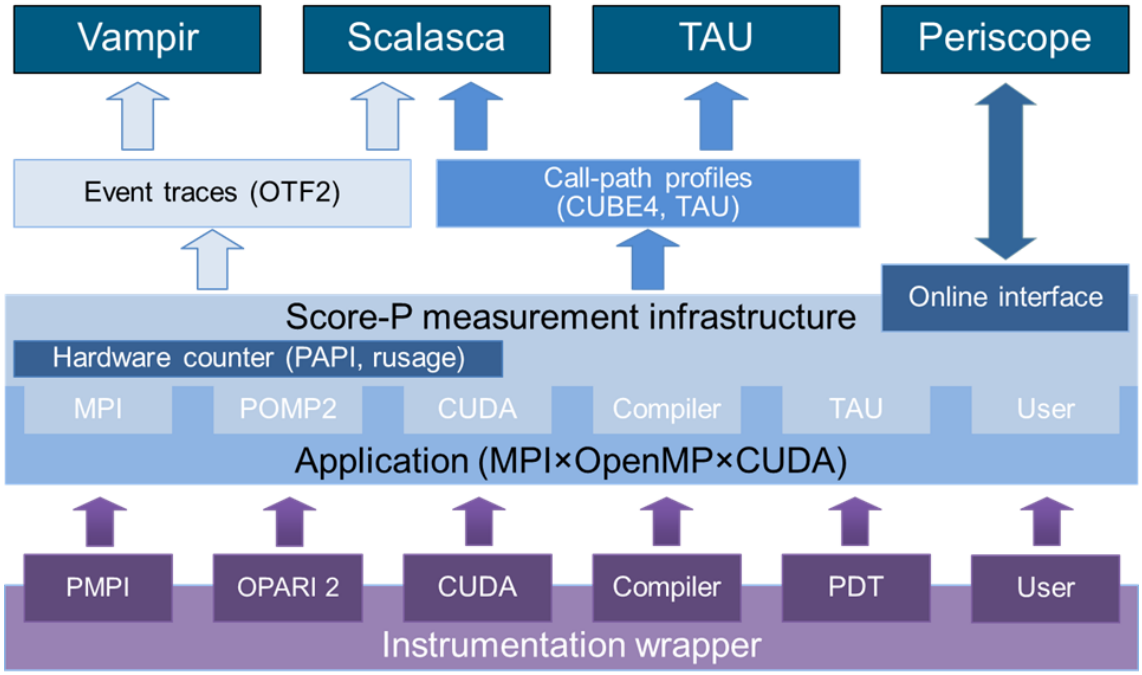

Figure 1 Architecture of the Score-P instrumentation and measurement system.

2.1 Performance Data and Topology Information

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.

Cray provides the PMI interface on its systems to gather hardware topology information like node, router asic information, blade, cage, and cabinet information. Score-P was extended to make use of this information and stores it internally within a tree-based system hierarchy (see Figure 2). This hierarchy information is needed to classify different threads of execution and also to associate machine and node performance counter with the application performance information.

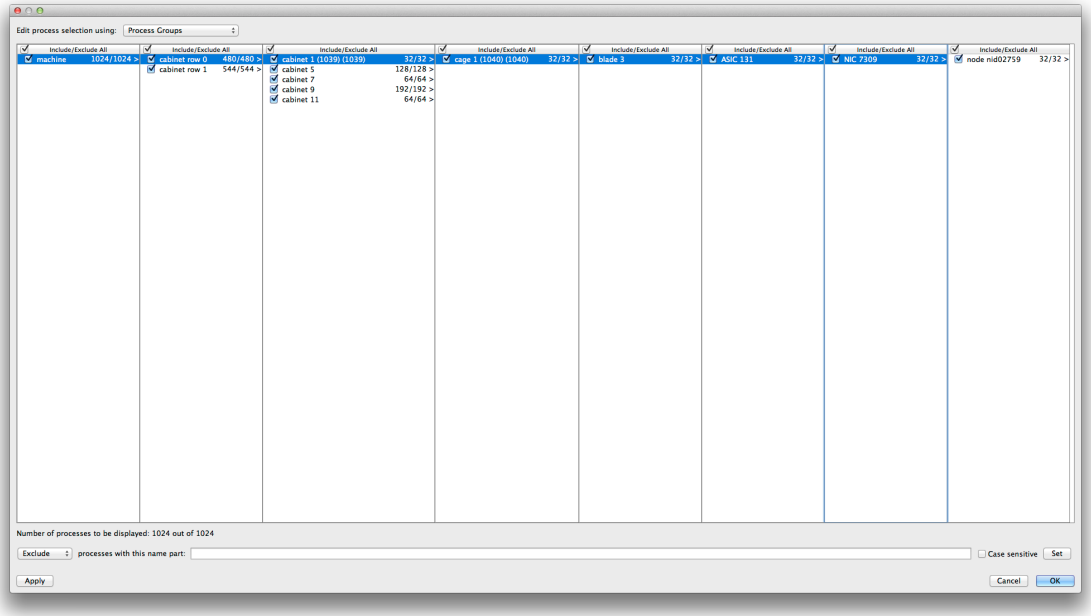

Figure 2 Visualisation of the Cray hardware topology within Vampir's filter dialog.

Since version 1.2 the measurement system is able to record generic and user-defined hierarchical performance counters with a metric plugins interface to address the level of complexity of today's machine architectures and those of the future. This allows us to add synchronously and asynchronously recorded performance values like energy or power to the application monitoring information. Figure 3 shows the visual representation of a MPI parallel application using hyper-threading, running on four nodes with 32 processes each, and corresponding node power and freshness information with Vampir.

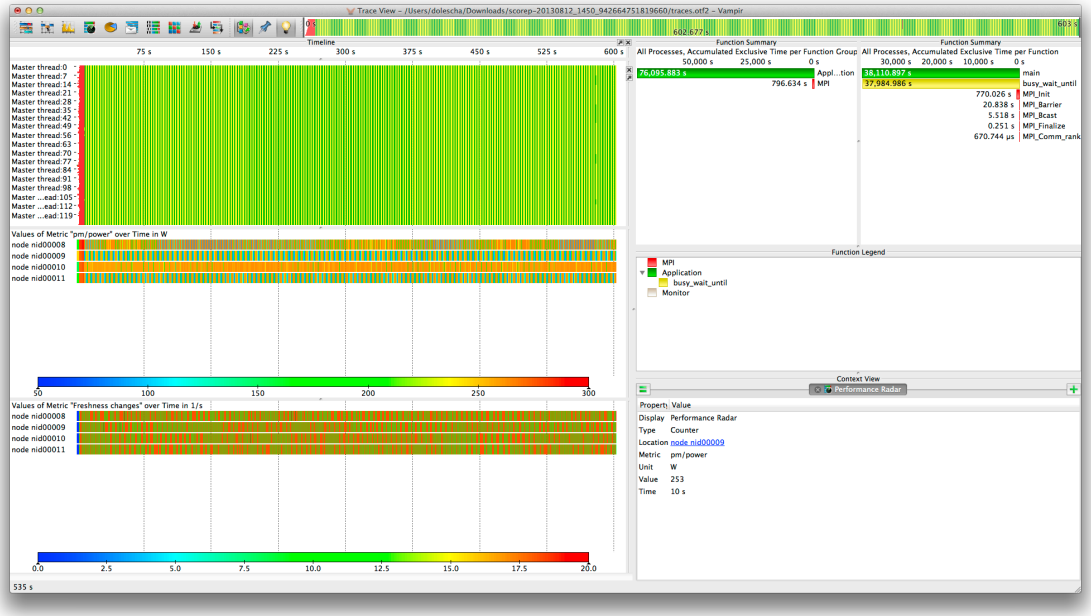

Figure 3 Performance visualisation with Vampir of the application behaviour and node power and freshness information of a MPI parallel application using hyper-threading, running on four nodes with 32 processes each monitored on a Cray XC30.

2.2 Scalability

The continuation of the current fine-grained operation mode would produce an enormous amount of data (GBytes to TBytes in a few seconds) 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. One prerequisite of dynamic filtering approaches is that they do not change the causal context. Especially, filtering of MPI point-to-point communication events leads classical matching algorithms to produce mismatching or failure. Therefore, we proposed a new fault tolerant matching scheme in [18] that produces reliable matching even in the presence of missing send or receive events.

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 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 Parallel Programming Models

To address the complexity and heterogeneity of exascale architectures new parallel programming models and 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 flexible task identification model within Score-P so that it is able to handle multiple parallel programming models and paradigms used simultaneously within one application.

2.3.2 Performance Monitoring of Hardware Accelerators (CUDA/OpenCL)

In the last years CUDA/OpenCL capable devices became more and more popular in the High Performance computing area since they are promising 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.

Since version 1.2 Score-P is able to monitor CUDA activities via CUPTI and OpenCL activities via a shared library wrapping approach. The use of the new developed generic one-sided RMA event model allows us to monitor memory transfers between host and graphic card as one-sided communication.

2.3.3 Performance Monitoring of PGAS Models

Partitioned Global Address Space (PGAS) models are available as library-based paradigms, e.g., Global Address Space Programming Interface (GASPI), SHMEM, as language extensions, e.g., UPC, Co-Array Fortran (CAF), and also new languages, e.g., Chapel.

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

1. If available by using a compiler instrumentation interface.
2. If available by using the monitoring interface provided by the PGAS model.
3. 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.
4. 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 one possibility is to extend a source-to-source instrumentor like OPARI to instrument these language extensions [9] or to convince the compiler vendors to provide language specific instrumentation interfaces.
5. Depending on the system implementation of these language extensions it is sometimes more applicable to monitor 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 Co-Array Fortran running on the same node.

To exchange data between the different memory locations PGAS languages use RMA (Remote Memory Access) operations as their underlying communication substrates. Therefore, we investigated one-sided communication models and developed a generic event model to record RMA operations in the OTF2 trace format for range of one-sided APIs and libraries [14]. Within CRESTA the Co-Array Fortran co-design team was established to investigate the possibilities and potentials of this PGAS language to overlap communication and computation within a world leading production application like ECMWF's Integrated Forecasting system (IFS). It turns out that the monitoring of

Cray's Co-Array Fortran fine granular operations will be only possible by using a source-to-source instrumentor or by indirect monitoring of the underlying communication library, i.e., monitoring of the Cray DMAPP library, due to the fact that the language constructs are processed in the compiler runtime. The same holds for the Cray UPC implementation.

Instrumentation of UPC application build with the GNU UPC or Berkeley UPC compiler can be done through the GASP tool interface [16]. Because of time constraints and no demand for UPC within CRESTA we will not investigate the monitoring via this interface in more detail.

Besides the monitoring of the Cray DMAPP library we also currently investigate the possibilities to monitor other PGAS models like GASPI and SHMEM, which are also promising to scale towards exascale.

2.3.3.1 Monitoring of the Cray DMAPP Library

On Cray systems Co-Array Fortran and UPC routines make use of the libpgas library, which uses the DMAPP library as underlying communication library. The calls to this library can be intercepted with a library wrapping approach and one-sided communication operations can be recorded with the generic one-sided RMA event model. Initialisation and finalisation with hierarchical unification can be done using MPI as underlying communication layer. Figure 4 shows the visualisation of a short Co-Array Fortran example. It can be observed that there are tiny functions, which are called very frequently like for example dmapp_c_pset_test. For tiny functions, which are called very frequently, it is advisable to disable the detailed monitoring and to enable only profiling to prevent the monitoring system to be swamped by these functions or in worst case if the overhead is too high to disable the monitoring of this class of functions.

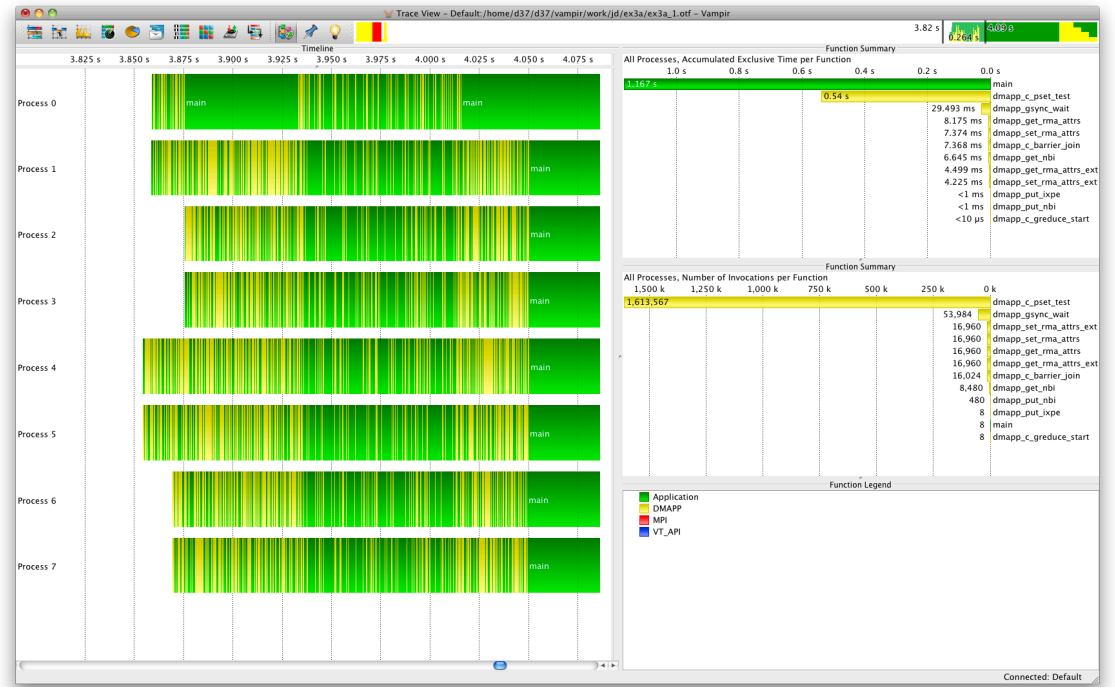

Figure 4 Performance visualisation of the Cray DMAPP communication library with Vampir. It is important to see that dmapp_c_pset_test is called very often and therefore should not be recorded in detail to reduce the overhead of the monitoring.

2.3.3.2 Monitoring of GASPI

The Global Address Space Programming Interface (GASPI) is a PGAS API based on asynchronous communication and execution model. GASPI provides configurable RDMA PGAS memory segments and allows application developers to map the memory heterogeneity of modern supercomputer to these PGAS segments. All these segments

can directly read and write from/to each other – within the node and across all nodes [17].

The data exchange in GASPI is based on one-sided asynchronous communication and can be therefore easily adapted to work with Score-P's generic RMA event model.

Since GASPI provides no point-to-point communication operation, which is a prerequisite of Score-P to map the local identifiers to global and unique identifiers within the hierarchical unification operation, the unification interface has to be adapted to the needs of GASPI. This can be achieved by implementing a hierarchical unification model based on the asynchronous GASPI communication operations or by emulation of point-to-point operations with asynchronous communication operations and reuse of the existing point-to-point hierarchical unification model.

The instrumentation of the GASPI library and therefore the generation of information can be done by using a library wrapping approach and definition of the event tracing interface.

Currently, GASPI is no object of the CRESTA co-design research but it is further promising asynchronous approach to overlap communication and computation and therefore might be considered and investigated in the future.

2.3.3.3 Monitoring of SHMEM

SHMEM is a PGAS paradigm very similar with MPI to pass data between cooperating parallel processes on logically shared distributed memory.

2.3.3.3.1 Monitoring of Cray SHMEM

The one-sided communication operations of Cray SHMEM can be easily recorded with Score-P's generic RMA event model.

Cray SHMEM allows the coexistence of MPI and therefore initialisation and finalisation of the measurement system can be easily used with the MPI interface of Score-P.

The instrumentation can either be done by a library wrapping approach or by definition of weak symbols and use of the strong symbols provided by the Cray SHMEM library, this approach is very similar to the PMPI interface of MPI.

Figure 5 shows the visualisation of the performance monitoring of a Cray SHMEM application.

2.3.3.3.2 Monitoring of OpenSHMEM

In 2010 the OpenSHEM community started an effort to standardize SHMEM and to bring together a variety of SHMEM and SHMEM-like implementations into an open standard [15].

Based on the one-sided communication API, the monitoring of these functions should work with the generic RMA event model of Score-P. But nevertheless, OpenSHMEM uses an own communication layer and integration of OpenSHMEM in Score-P is more time consuming than other PGAS implementation like Cray SHMEM. Since there is at the moment no demand for OpenSHMEM within CRESTA and also there are time constraints, we will not investigate the monitoring of OpenSHMEM in more detail.

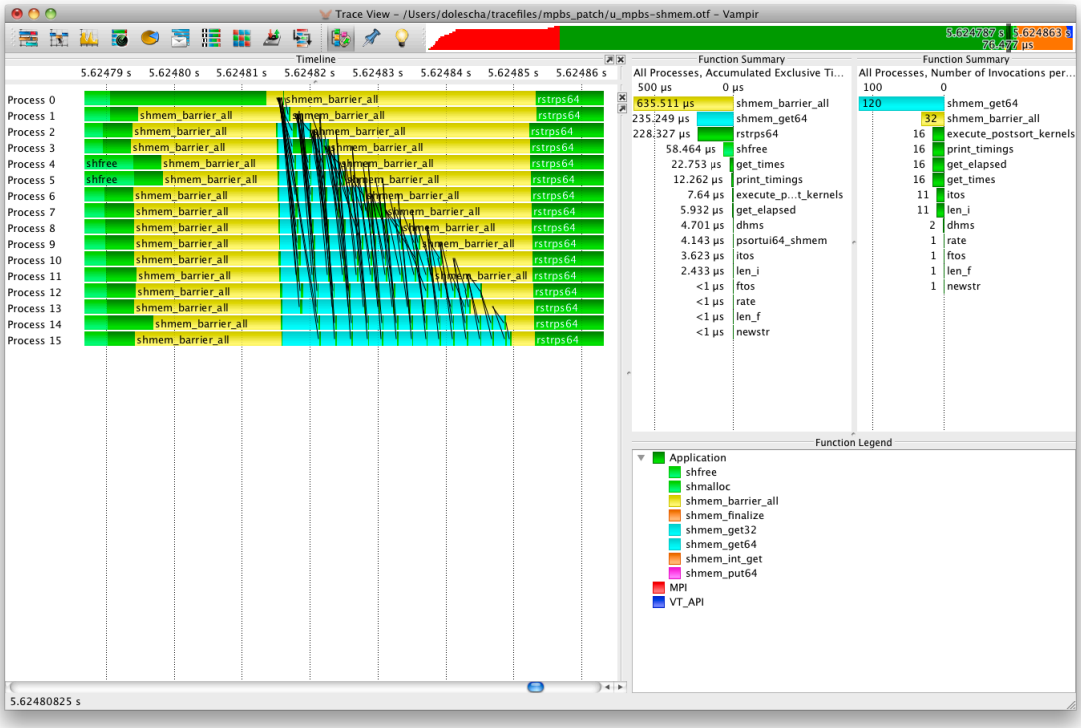

Figure 5 Performance visualisation of a massive parallel bucket sort parallelized with Cray SHMEM. The master timeline shows very impressively the master-slave communication implemented with shmem_get64 operations coloured in light blue between the different processes surrounded by tow shmem_barrier_all operations coloured in yellow.

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 Possible Future Extensions

1. Extension of Score-P to support library wrapping of shared libraries.
2. Extension of Score-P Pthreads, and maybe further parallel programming models.
3. Flexible and generic task identification model that supports multiple parallel paradigms together.
4. 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 million 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, and 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. Vampir uses pre-calculated summary information to partially load and analyse large event trace information and to visualise only specific segments of the whole monitoring run. In addition it also allows to exclude specific threads of execution from the analysis and visualisation, see Figure 6 as an example for this feature while loading a NEK5000 trace file.

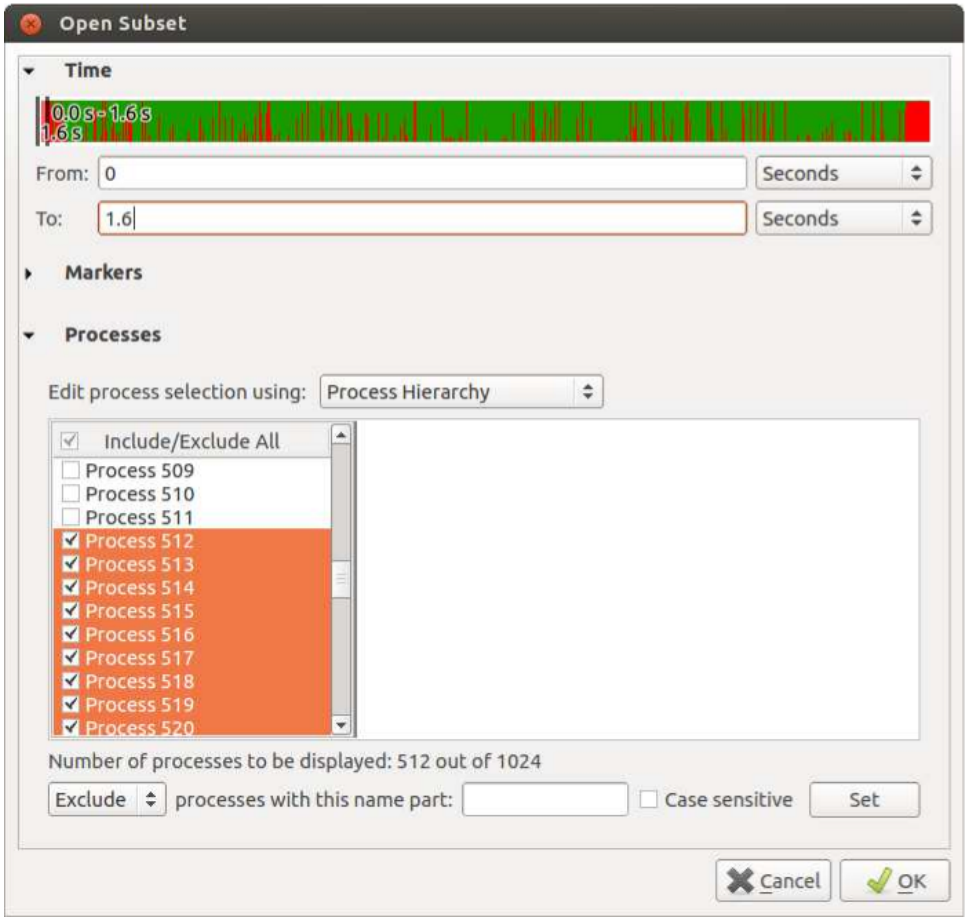

Figure 6 Vampir's Partial Loading Dialog showing the loading of the NEK5000 trace in the time range of 0 seconds to 1.6 seconds for the processes 512-1024.

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 Parallel Programming Models

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, Vampir allows to explicitly select and deselect specific threads of execution by their paradigm, their location in the system tree, and also by their name, see Figure 7 and Figure 8. The information displayed in the various timeline and statistic displays will be adapted to this selection rules.

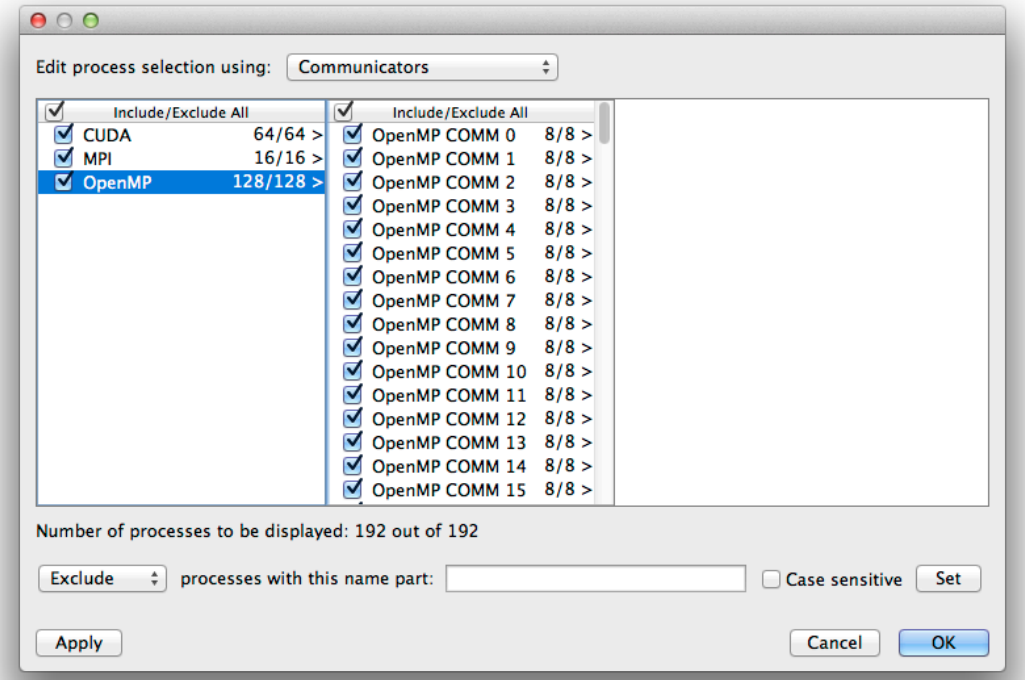

Figure 7 Vampir's filtering dialog to filter threads of execution based on the paradigm.

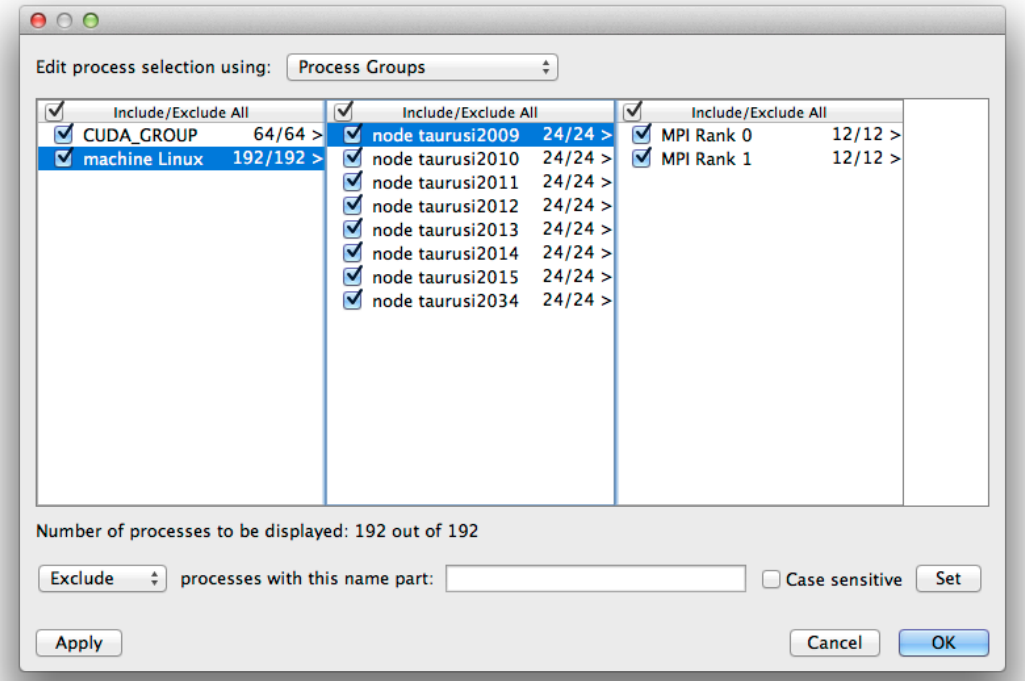

Figure 8 Vampir's filtering dialog to filter threads of execution by their topology information.

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. In addition we test our tools with their functionality and features with the CRESTA applications Gromacs, IFS, Nek5000, HemeLB, and OpenFoam.

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.

4.3 Performance Monitoring Tests and Performance Visualisation Results of CRESTA Applications

This section provides a short summary of the monitoring tests and performance visualisation with Vampir of each individual CRESTA application. The monitoring results were created on the Cray machines provided by KTH, HLRS, and KTH. The number of the used threads of execution mainly depends on the provided input files and varies from 128 to 2040 cores.

The intention of this section is to demonstrate and to test the usability of the performance tools and not to provide a detailed performance issues or scalability discussion. Each application group should address this individually.

4.3.1 Performance Monitoring Test and Visualisation Result of Gromacs

We monitored Gromacs with various numbers of threads of execution and different parallel paradigms from pure MPI applications to hybrid versions of Gromacs MPI+OpenMP see Figure 10, or MPI+OpenMP+CUDA see Figure 9.

Since Gromacs uses beside a data parallelism also a function parallelism, the use of Vampir's filter functionality helps to investigate different classes of parallelism in more detail see Figure 10, Figure 11, and Figure 12.

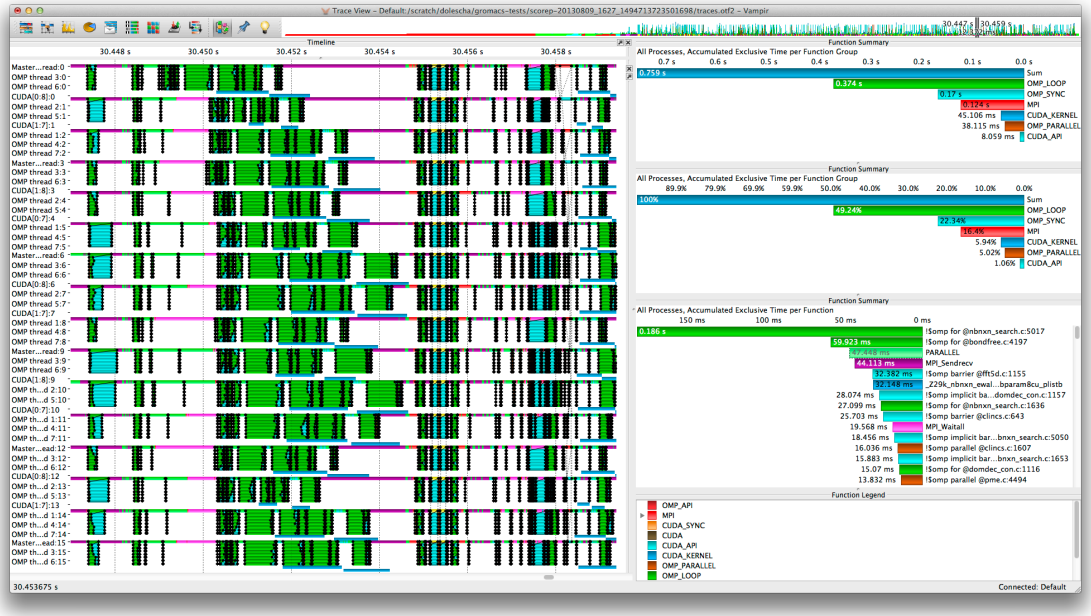

Figure 9 Performance visualisation of a hybrid version of Gromacs parallelized with MPI, OpenMP, and CUDA running on 8 nodes with 16 cores and 2 Nvidia K20 each. Every process uses one Nvidia graphic card and sends its kernels coloured in blue to two different streams.

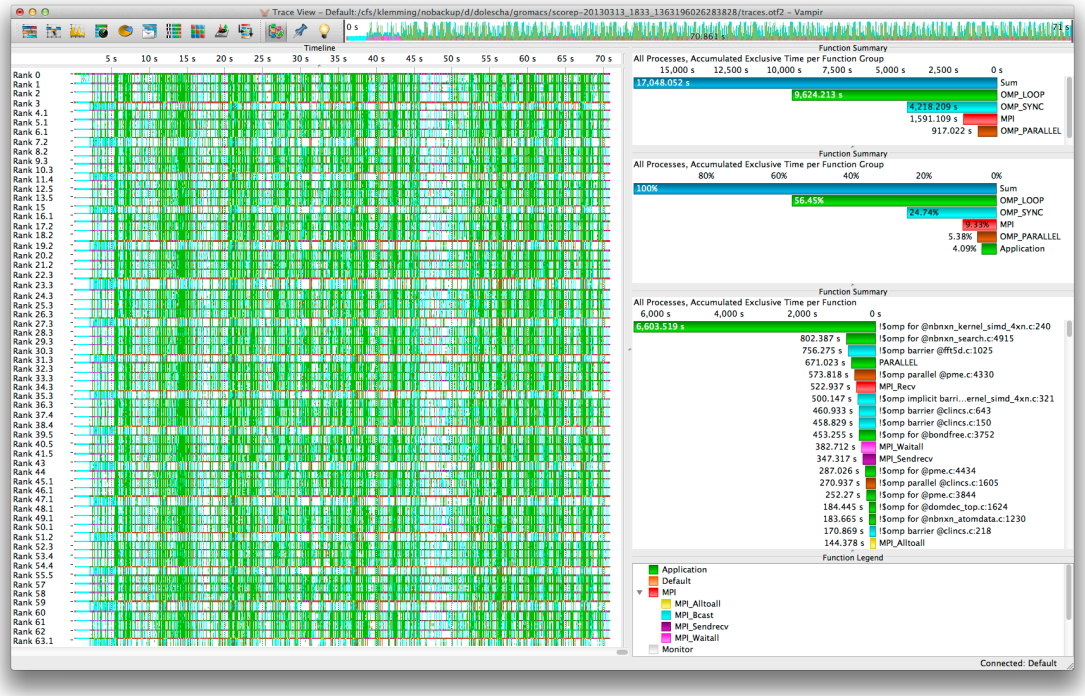

Figure 10 Performance visualisation with Vampir of the runtime behaviour of a hybrid Gromacs parallelized with MPI and OpenMP. One quarter of the processes and corresponding threads are responsible to compute the bonded interactions and integration (PME).

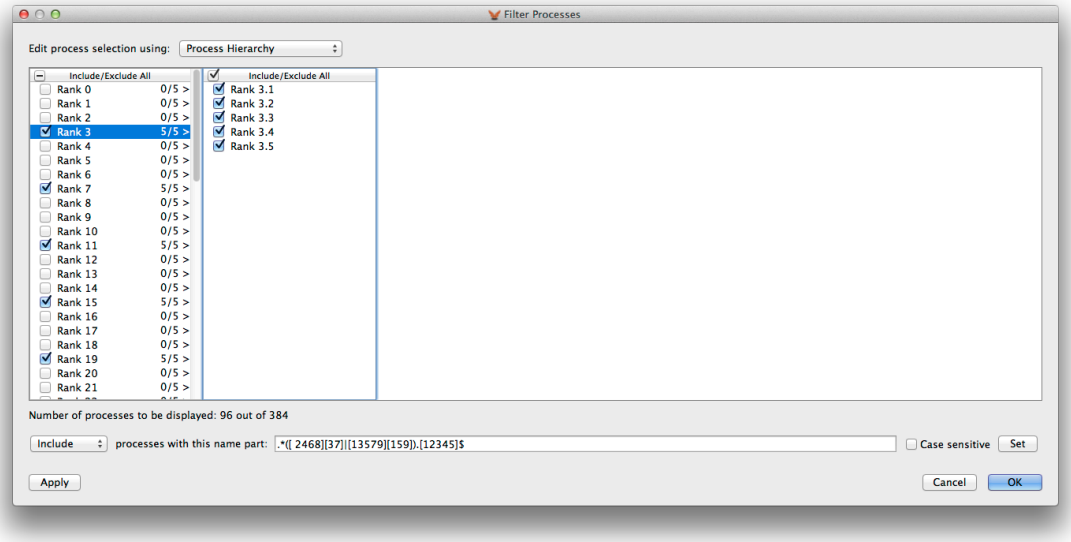

Figure 11 Vampir's filter dialog to select only the PME processes by using a regular expression. The resulting visualisation can be seen in Figure 12.

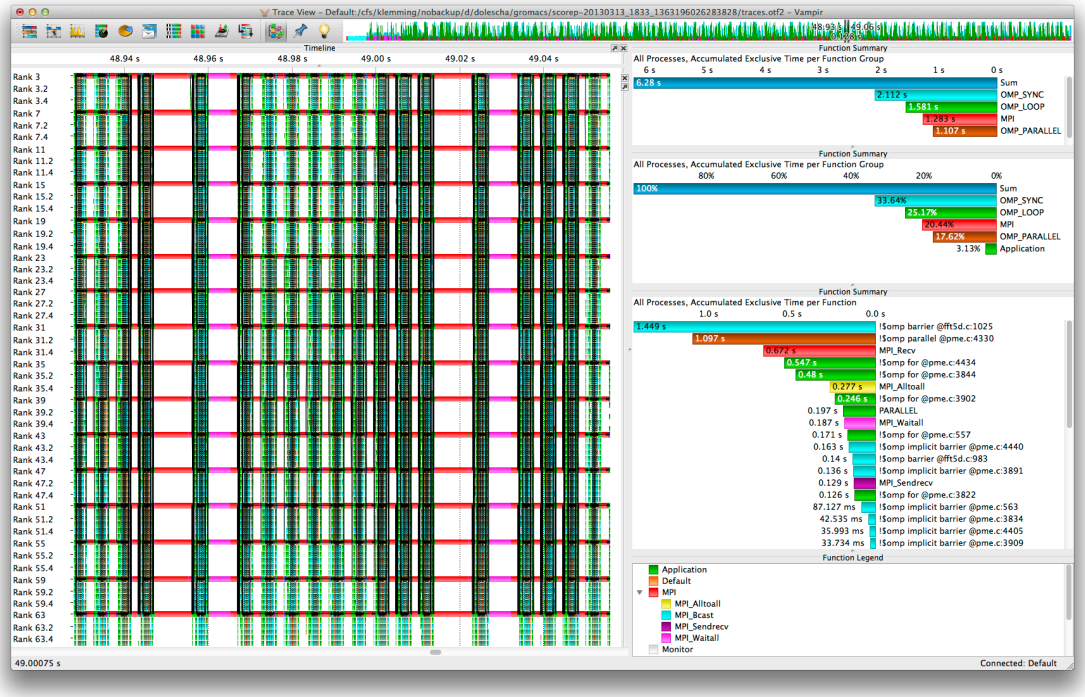

Figure 12 Performance visualisation of a selection of processes and threads (PME) with Vampir for a more detailed and focussed performance analysis without any "colouring noise" of the rest of the processes and threads.

4.3.2 Performance Monitoring Test and Visualisation Result of IFS

We monitored a hybrid version of IFS parallelized with MPI and OpenMP running with dataset T1023. The performance visualisation of the application behaviour with Vampir can be seen in Figure 13.

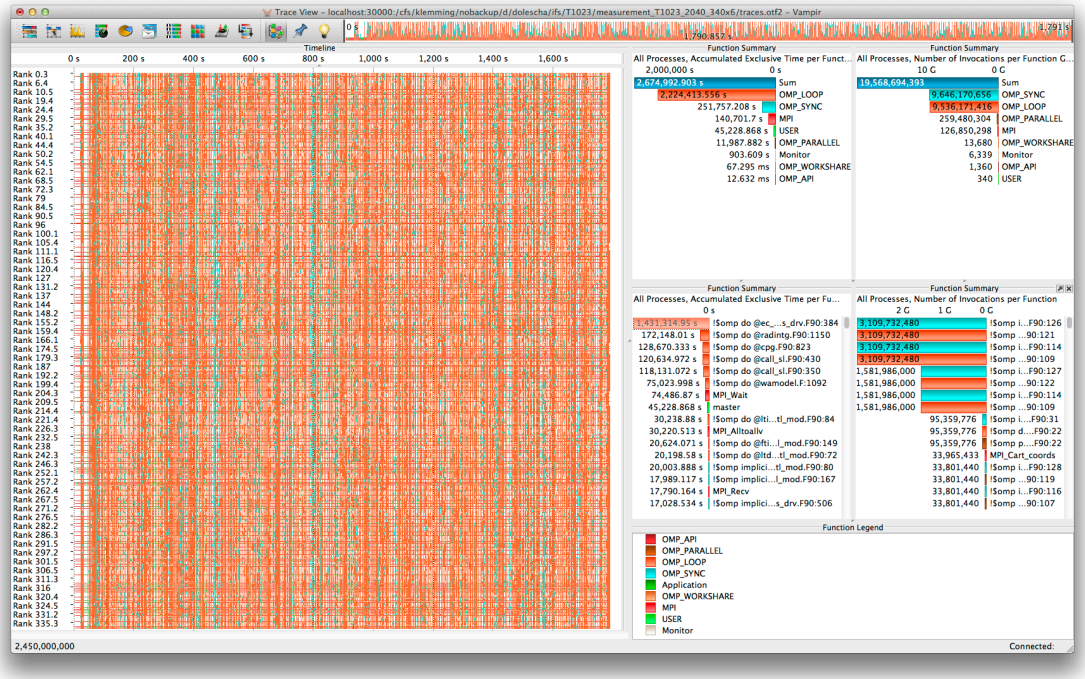

Figure 13 Performance visualisation with Vampir of a hybrid IFS T1023 run.

4.3.3 Performance Monitoring Test and Visualisation Result of Nek5000

We monitored a MPI parallel version of Nek5000 with a jet data input set. The performance visualisation with Vampir can be seen in Figure 14.

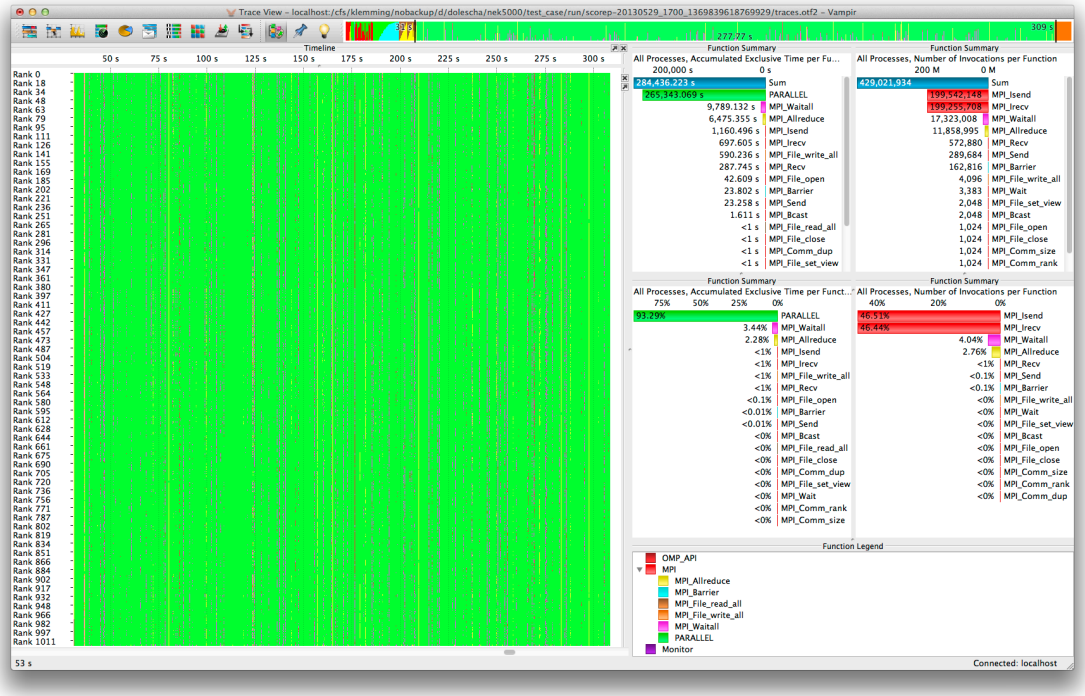

Figure 14 Performance visualisation with Vampir of Nek5000 parallelised with MPI.

4.3.4 Performance Monitoring Test and Visualisation Result of HemeLB

We monitored a MPI parallel version of HemeLB. The performance visualisation with Vampir can be seen in Figure 15.

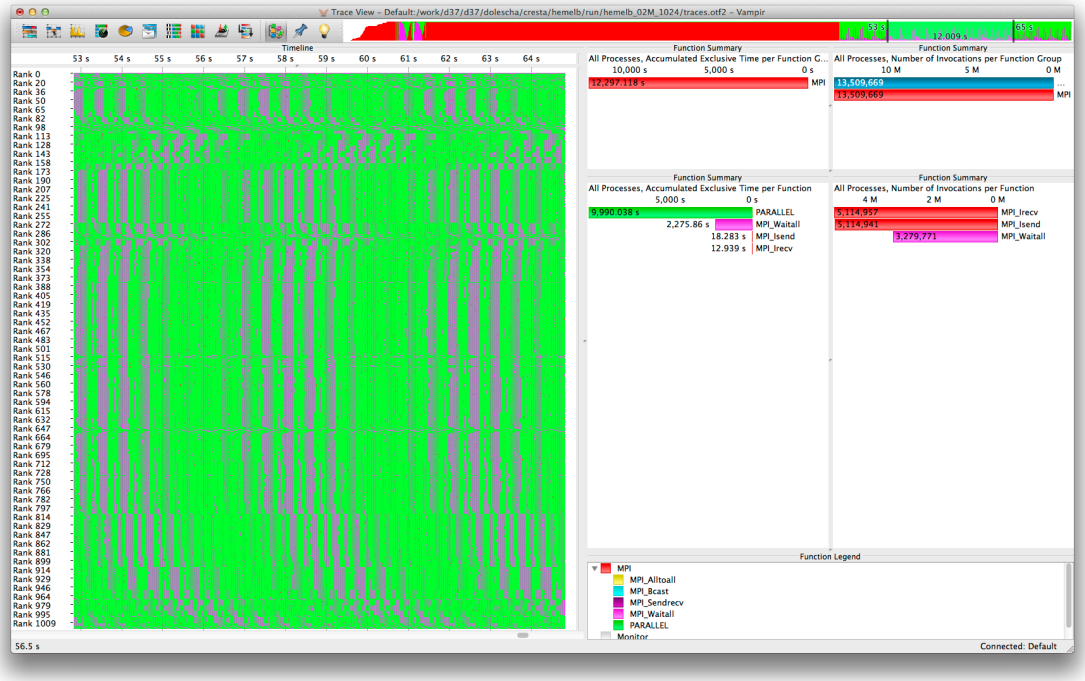

Figure 15 Performance visualisation with Vampir of HemeLB parallelised with MPI.

4.3.5 Performance Monitoring Test and Visualisation Result of OpenFoam

We monitored a MPI parallel version of OpenFoam. The performance visualisation with Vampir can be seen in Figure 16.

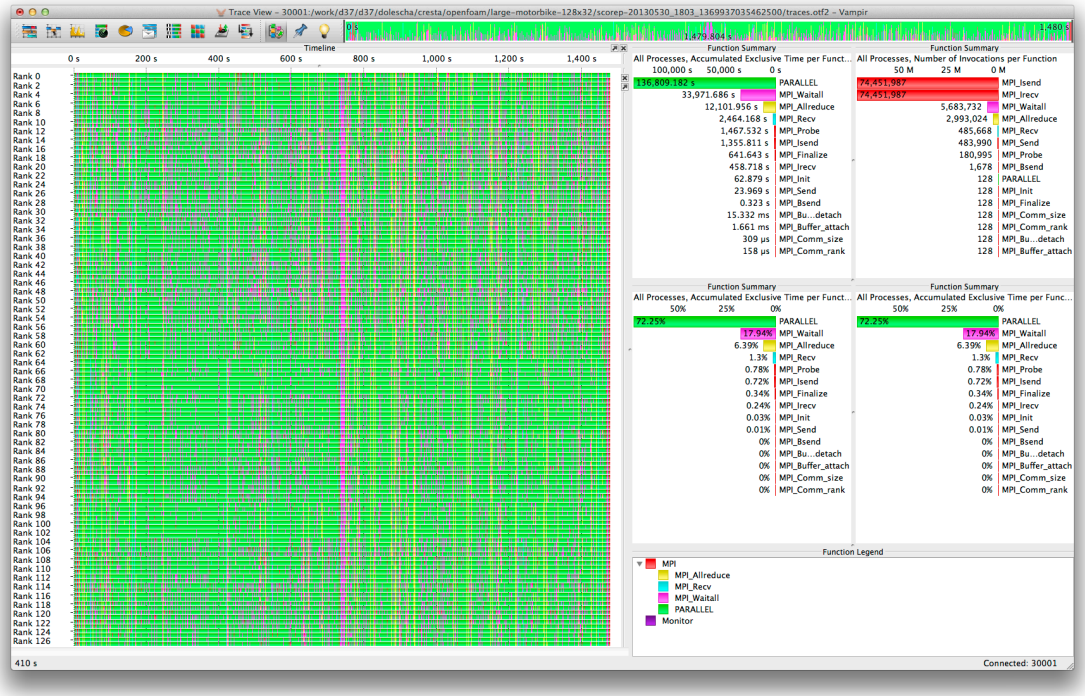

Figure 16 Performance visualisation with Vampir of OpenFoam parallelised with MPI.

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.

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.

[14] Knüpfer, A., Dietrich, R., Doleschal, J., Geimer, M., Herrmans, M.-A., Rössel, C., Tschüter, R., Wesarg, B., Wolf, F.: Generic Support for Remote Memory

Access Operations in Score-P and OTF2, Tools for High Performance Computing 2012, Springer Berlin Heidelberg, pp. 57-74, 2013

[15] Chapman, B., Curtis, T., Pophale, S. Poole, S., Kuehn, J., Koelbel, C., Smith, L.: Introducing OpenSHMEM: SHMEM for the PGAS community, PGAS'10 Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model, ACM New York, NY, USA, 2010

[16] Su, H.-H., Bonachea, D., Leko, A., Sherburne, H., Billingsley, M., George, A. D.: GASP! a standardized performance analysis tool interface for global address space programming models. In: Proceedings of the 8th international con- ference on Applied parallel computing: state of the art in scientific computing. Berlin, Heidelberg : Springer-Verlag, 2007 (PARA'06). – ISBN 3–540–75754–6, 978–3–540– 75754–2, S. 450–459

[17] Alrutz, T. and Backhaus, J. and Brandes, T. and End, V. and Gerhold, T. and Geiger, A. and Grünewald, D. and Heuveline, V. and Jägersküpper, J. and Knüpfer, A. and Krzikalla, O. and Kügeler, E. and Lojewski, C. and Lonsdale, G. and Müller-Pfefferkorn, R. and Nagel, W. and Oden, L. and Pfreundt, F.-J. and Rahn, M. and Sattler, M. and Schmidtobreick, M. and Schiller, A.and Simmendinger, C. and Soddemann, T. and Sutmann, G. and Weber, H. and Weiss, J.-P.: GASPI at Multi-Core Challenge III, in R. Keller et al. (Eds.): Facing the Multicore-Challenge III 2012, LNCS 7686, pp. 135--136. Springer, Heidelberg (2013)

[18] Wagner, M., Doleschal, J., Knüpfer, A., and Nagel, W.E.: Runtime Message Uniquification for Accurate Communication Analysis on Incomplete MPI Event Traces**.** In: Proceedings of the 20th European MPI Users's Group Meeting, 2013

