D3.9 – Final Release of Performance Analysis Tools

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 Exa-scale computing, software and simulation

Due date:	M38
Submission date:	30/11/2014
Project start date:	01/10/2011
Project duration:	39 months
Deliverable lead organization	KTH
Version:	1.0
Status	Final version
Author(s):	Michael Wagner (TUD), Jens Doleschal (TUD)
Reviewer(s)	Luis Cebamanos (EPCC), Derek Groen (UCL)

Dissemination level	
PU	*PU*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	14/11/2014	First draft version	Michael Wagner (TUD), Jens Doleschal (TUD)
0.2	17/11/2014	Review comments	Luis Cebamanos (EPCC), Derek Groen (UCL)
0.3	24/11/2014	Revised version	Michael Wagner (TUD), Jens Doleschal (TUD)
1.0	27/11/2014	Final version for submission	Catherine Inglis (EPCC)

Table of Contents

1 Executive Summary

This deliverable reports on the availability of, and access to, the final versions of the performance analysis tools developed in WP3 of the CRESTA project. This includes two tools:

- Score-P and
- Vampir

Score-P is a highly scalable tool to monitor parallel applications. It supports a wide range of programming languages and parallel programming paradigms. In addition, Score-P provides different monitoring modes namely profiling, tracing, and online monitoring.

Vampir is a graphical tool to visualize and analyze applications monitored with Score-P in post-mortem. It provides different displays and techniques to visualize the details of highly parallel applications in a scalable and user-friendly way.

2 Introduction

We report on the availability of both Score-P and Vampir in Section 3. The general usage of both Score-P and Vampir is documented in the Score-P and Vampir user manual contained in the software distribution. This section covers functionality that is not or only partly covered in the Score-P user manual and is of specific interest for the CRESTA project.

2.1 Purpose

The purpose of this deliverable is to:

- Provide access information for the performance visualizer Vampir
- Provide access information for the performance monitor Score-P
- Provide usage information for the performance monitor Score-P

3 Performance Analysis Tools

This section describes how to obtain the tools used in the CRESTA project and how to use the new features.

3.1 Accessing the Software Packages

3.1.1 Score-P

The Score-P measurement infrastructure [1] is a highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis of HPC applications. Score-P is distributed under a BSD-License and can be obtained at:

http://www.vi-hps.org/projects/score-p/

Score-P 1.4 will be released soon and Score-P 1.4 alpha is available from the CRESTA repository with the name **scorep-1.4-alpha.tar.gz**.

3.1.2 Plugins for Energy and Network Information Monitoring

To monitor energy and network counter information on Cray XC platforms, Score-P provides a plugin interface to collect this data and to correlate them with the application information [2]. For this purpose there are two plugins available from the CRESTA repository as follows:

- Energy and power monitoring: **pm_plugin.tar**
- Network counter monitoring: **apapi.tar**

3.1.3 Vampir

Vampir 8.3 [3] focuses on extending the graphical presentation of performance data and adds support for the collaborative Score-P performance monitor release version 1.2. Compatibility with earlier OTF and VampirTrace releases is maintained. New feature highlights include:

- Hierarchical process folding in the master timeline.
- Introduction of combinable peer-to-peer communication metrics in the performance radar.
- Complete revision of comparison and alignment mode for multiple traces with session buffering support and a session manager.
- Pre-selection of processes or threads prior to loading performance data.
- A kiviat chart mode in the process summary chart.
- Quick access to color settings with support for unique or random color schemes.

Furthermore, various features and performance improvements, scalability and stability enhancements have been incorporated.

Vampir is distributed as commercial software. A demo version can be obtained at:

http://www.vampir.eu/

3.2 Tracing New Paradigms, Energy and Network Information

This section covers approaches to monitoring and analyzing new parallel paradigms and system metrics such as energy and network information.

3.2.1 Tracing OpenACC Usage

In the last few years, CUDA/OpenACC-capable devices have become more and more popular in the High Performance Computing area since they are promising more floating point operations per second than a typical CPU will ever provide in a user application.

Host-side activities of OpenACC-capable devices can be monitored either 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 either be done via the CUDA Profiling Tools Interface (CUPTI) or by the previously mentioned library wrapping approach. CUPTI provides different APIs that can be used to get insight into the CPU and GPU behavior 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 1.3, Score-P has been able to monitor CUDA activities via CUPTI and OpenACC activities via a shared library wrapping approach. The use of the newly-developed generic one-sided RMA event model allows us to monitor memory transfers between host and graphic card as one-sided communication. To enable the monitoring of these events the application has to be linked against the monitoring library and the following runtime environment variables must be set:

```
SCOREP_CUDA_ENABLE=kernel,memcpy,driver,concurrent
SCOREP_CUDA_BUFFER=3M
```

3.2.2 Tracing Energy and Power Information

Energy and power consumption are increasingly important topics in High Performance Computing. Delivering sustained but energy-efficient performance of real-world applications will require software engineering decisions, both at the systemware level but also in the applications themselves. Such application decisions might be made when the software is designed or at runtime via an auto-tuning framework.

For these to be possible, fine-grained instrumentation is needed to measure energy and power usage not just of overall HPC systems but also of individual components within the architecture. This information also needs to be accessible not just to privileged system administrators but also to individual users of the system, and in a way that is easily correlated with the execution of their applications.

Score-P has been able to record external generic and user-defined hierarchical performance counters since version 1.2. This is done with a flexible “metric plugins” interface to address the complexity of machine architectures both today and in the future. The metric plugin interface provides an easy way to extend the core functionality of Score-P to record additional counters, which can be defined in external libraries and loaded at application runtime by the measurement system. We built a Score-P metric plugin to monitor application-external energy and power information on Cray platforms during the application measurement to run asynchronously per node [1].

To use the power monitoring plugin it must be built on the target system and the application must be instrumented at the desired level of detail. Setting the according environment variables activates this power monitoring plugin:

```
export SCOREP_METRIC_PLUGINS=pm_plugin
export SCOREP_METRIC_PM_PLUGIN=”all”
```

3.2.3 Tracing of Network Information

With systems getting larger and more complex, networks within HPC systems are getting more and more complex as well. Since network problems or high network load can tremendously affect the behavior of parallel applications it is important to enable an analysis of the correlations between network and application behavior.

Similar to external energy counters, network statistics and counters can be monitored and integrated in an application trace with the Score-P metric plugin interface by using

an according plugin that calls PAPI interface asynchronously per node. In addition, the according environment variables must be set. However, the available counters may vary on each platform:

```
export SCOREP_METRIC_PLUGINS=APAPI
export \
    SCOREP_METRIC_APAPI="AR_NIC_NETMON_ORB_EVENT_CNTR_REQ_STALLED,\
    AR_NIC_NETMON_ORB_EVENT_CNTR_RSP_STALLED,\
    AR_NIC_NETMON_ORB_EVENT_CNTR_REQ_PKTS,\
    AR_NIC_NETMON_ORB_EVENT_CNTR_RSP_PKTS,\
    AR_NIC_NETMON_ORB_EVENT_CNTR_REQ_FLITS,\
    AR_NIC_NETMON_ORB_EVENT_CNTR_RSP_FLITS"
```

3.3 Selective Monitoring

Event tracing tools record each event of a parallel application in detail. Thus, it allows the dynamic interaction between thousands of concurrent processing elements to be captured and enables the identification of outliers from the regular behavior. While single events are rather small, event-based tracing frequently results in huge data volumes. We developed and evaluated three approaches to address the large amount of collected data, in particular for massively parallel or long-running applications. First, using different levels of detail by enabling or disabling certain parallel paradigms or preventing the instrumentation of functions that are usually inlined by the compiler. Second, applying a rewind within the record event stream to subsequently remove iterations that are not of interest and only keep those that represent deviating behavior. Third, removing highly frequent short-running function calls that can overwhelm any recording memory buffer while at the same time contributing very little to the analysis and understanding of the overall application behavior.

3.3.1 Monitoring Different Levels of Details for Each Process

To compare different levels of details it is possible to build different instrumented versions of an application. For a multi-paradigm application like Gromacs this could be:

- Compiler instrumentation + MPI + OpenMP + CUDA,
- Compiler instrumentation with filters + MPI + OpenMP + CUDA,
- MPI + OpenMP + CUDA, or
- MPI + CUDA.

This can be achieved by setting the according instrumentation options in Score-P:

```
scorep --mpp=mpi --thread=omp:pomp_tpd
scorep --mpp=mpi --thread=omp:pomp_tpd --filter=<file>
scorep --mpp=mpi --thread=omp:pomp_tpd --nocompiler
scorep --mpp=mpi --thread=none --nocompiler
```

Currently the minimal instrumentation must contain MPI to get an entry point with MPI_Init and MPI_Finalize. In the future a wrapper that intercepts only MPI_Init and MPI_Finalize would reduce the minimal instrumentation further.

You can use aprun to launch the differently instrumented application in MPMD mode. Shell scripts can be used to set different environments for each version:

```
aprun -n pes [aprun_options] executable1 [args_ executable1] : \
      -n pes [aprun_options] executable2 [args_ executable2] : \
      -n pes [aprun_options] executable3 [args_ executable3]

aprun -n 12 ./app1 : -n 8 ./app2 : -n 32 ./app3
```

3.3.2 Selective Monitoring of Iterations

Selective monitoring is one approach to decreasing the number of collected events. There are two main methods to select the recorded events: static and dynamic selection. For example, in iterative applications it is reasonable to avoid storing every single iteration, because most of them show more or less the same behavior. Therefore, the first method is to statically define which iteration is recorded and stored, e.g., every 10th or 100th iteration. With this it is still possible to analyze the behavior over time but the amount of recorded data is reduced to ten or one percent, respectively. However, iterations with either interesting behavior or a performance problem might be lost. The second method is to record every iteration and dynamically decide whether it is stored or discarded by evaluating its behavior, e.g. only store an iteration when its runtime varies from the average runtime by a defined offset. To realize such a subsequent removal of iterations we developed and applied a rewind method to rewind the recorded event stream to any pre-defined point (e.g. the beginning of the current iteration), which eliminates every record after that point [4].

3.4 Scalability

Event tracing delivers most detailed information allowing a profound post-mortem analysis of the parallel behavior. However, this comes with the cost of very large data volumes. Handling such a tremendous amount of data has always been a challenge in event tracing and is getting even more demanding with the rapid increase of processing elements. Since the collected data is traditionally stored in one file per processing element, the rising number of resulting event trace files is, in particular, one of the most urgent challenges. The limits of current parallel file systems allow handling of no more than around ten or twenty thousand parallel processes without any special treatment.

Writing one file per processing elements (e.g. check points or result files) does not scale to large systems since the sheer number of files overcharges the capabilities of today's file system meta-data servers. Score-P uses SIONlib [5], which relies on the file system's capability to handle large sparse files to pre-allocate segments for the logical file handles within a single file.

Since version 1.0, Score-P has supported the usage of SIONlib but has been restricted to pure MPI applications. With the upcoming release, Score-P 1.4 will support hybrid programs as well.

4 References

[1] A. Knüpfer, C. Rössel, D. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf: “Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir”, Tools for High Performance Computing 2011, Springer, pp. 79–91, 2012.

[2] A. Hart, H. Richardson, J. Doleschal, T. Ilsche, M. Bielert and M. Kappel: “User-level Power Monitoring and Application Performance on Cray XC30 Supercomputers”, Cray User Group Meeting 2014.

[3] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel: “The Vampir Performance Analysis Tool Set,” Tools for High Performance Computing, Springer, pp. 139–155, 2007.

[4] M. Wagner, J. Doleschal, A. Knüpfer and W. E. Nagel: “Runtime Message Uniquification for Accurate Communication Analysis on Incomplete MPI Event Traces”, Proceedings of the 20th European MPI Users' Group Meeting, Madrid, Spain, pages 123-128, ACM, 2013.

[5] W. Frings, F. Wolf, and V. Petkov: “Scalable massively parallel i/o to task-local files”, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ser. SC ’09, New York, NY, USA ACM, pages 17:1–17:11, 2009.