HOPSA Project

Technical Report

Automatic Performance Screening and Integrated Performance Analysis Tools Suite

September 2012

Felix Wolf[1], Judit Gimenez Lucas[2], Erik Hagersten[3], Thomas Ilsche[4], Andreas Knüpfer[4], Bernd Mohr[5], Harald Servat Gelabert[2], Aamer Shah[1], Zoltan Szebenyi[1]

1. German Research School for Simulation Sciences GmbH
2. Barcelona Supercomputing Center
3. Rogue Wave Software AB
4. Technical Universität Dresden
5. Forschungszentrum Jülich

HOlistic Performance System Analysis-EU

Deliverable D3.2

Workflow Report

CONTRACT NO	HOPSA-EU 277463
INSTRUMENT	CP (Collaborative project)
CALL	FP7-ICT-2011-EU-Russia

Due date of deliverable: May 15th, 2012

Actual submission date: June 15th, 2012

Start date of project: 1 FEBRUARY 2011 Duration: 24 months

Name of lead contractor for this deliverable: GRS

Name of reviewers for this deliverable: Felix Wolf, Bernd Mohr

Abstract: This deliverable specifies the overall performance-analysis workflow of the HOPSA environment. It describes all steps ranging from mandatory system-wide performance screening to the application of specialized diagnostic tools.

Revision 1.0

Project co-funded by the European Commission within the Seventh Framework Programme (FP7/2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Glossary

Abbreviation / acronym	Description
API	Application Programming Interface
Clustrx®	An operating system for high-performance computing from Massive Solutions. In this document, the term is used to refer to Clustrx Watch, the associated monitoring, management and control system.
GUI	Graphical User Interface
HOPSA	HOlistic Performance System Analysis
HPC	High Performance Computing
I/O	Input/Output
LWM^2	Lightweight Measurement Module
MPI	Message Passing Interface (Programming Model for Distributed Memory Systems)
OpenMP	Open Multi-Processing (Programming Model for Shared Memory Systems)
Score-P	A unified performance measurement infrastructure for parallel programs

1. Executive summary

This document describes the performance-analysis workflow defined in Task 3.2 of Work Package 3 of the EU FP7 project HOPSA. The HOPSA project (HOlistic Performance System Analysis) sets out for the first time to develop an integrated diagnostic infrastructure for combined application and system tuning. The document guides application developers in the process of tuning and optimising their codes for performance. It describes which tools should be used in which order to accomplish common performance analysis tasks. Since the document addresses primarily the user's perspective, it follows the style of a user guide. It does, however, not replace the user guides of individual performance-analysis tools developed in HOPSA but rather connects them as it shows how to use the tools in a complementary way. At the centre of this document is the so-called lightweight measurement module (LWM^2). Being responsible for the first step in the workflow, the system-wide mandatory collection of basic performance data, the module is covered in greater detail. Special emphasis is given to the interpretation of the job digest created with the help of LWM^2. The metrics listed in this compact report indicate whether an application suffers from an inherent performance problem or whether application interference may have been at the root of dissatisfactory behaviour. They also provide a first assessment regarding the nature of a potential performance problem and help to decide on further diagnostic steps using any of the more powerful performance-analysis tools. For each of those tools, a short summary is given with information on the most important questions it can help to answer. Moreover, the document covers Score-P, a common measurement infrastructure shared by some of the tools. The performance data types supported by Score-P form a natural refinement hierarchy that can be followed to track down and represent even complex bottleneck situations at increasing levels of granularity. Finally, a brief excursion on system tuning explains how system providers can leverage the data collected by LWM^2 to identify a suboptimal system configuration or faulty components.

2. Introduction

This document describes the performance-analysis workflow defined in Task 3.2 of Work Package 3 of the EU FP7 project HOPSA. The document guides application developers in the process of tuning and optimising their codes for performance. It describes which tools should be used in which order to accomplish common performance analysis tasks. Since the document addresses primarily the user's perspective, it follows the style of a user guide.

2.1 The broader context: The HOPSA project

To maximise the scientific and commercial output of a high-performance computing system, different stakeholders pursue different strategies. While individual application developers are trying to shorten the time to solution by optimising their codes, system administrators are tuning the configuration of the overall system to increase its throughput. Yet, the complexity of today's machines with their strong interrelationship between application and system performance demands an integration of application and system programming.

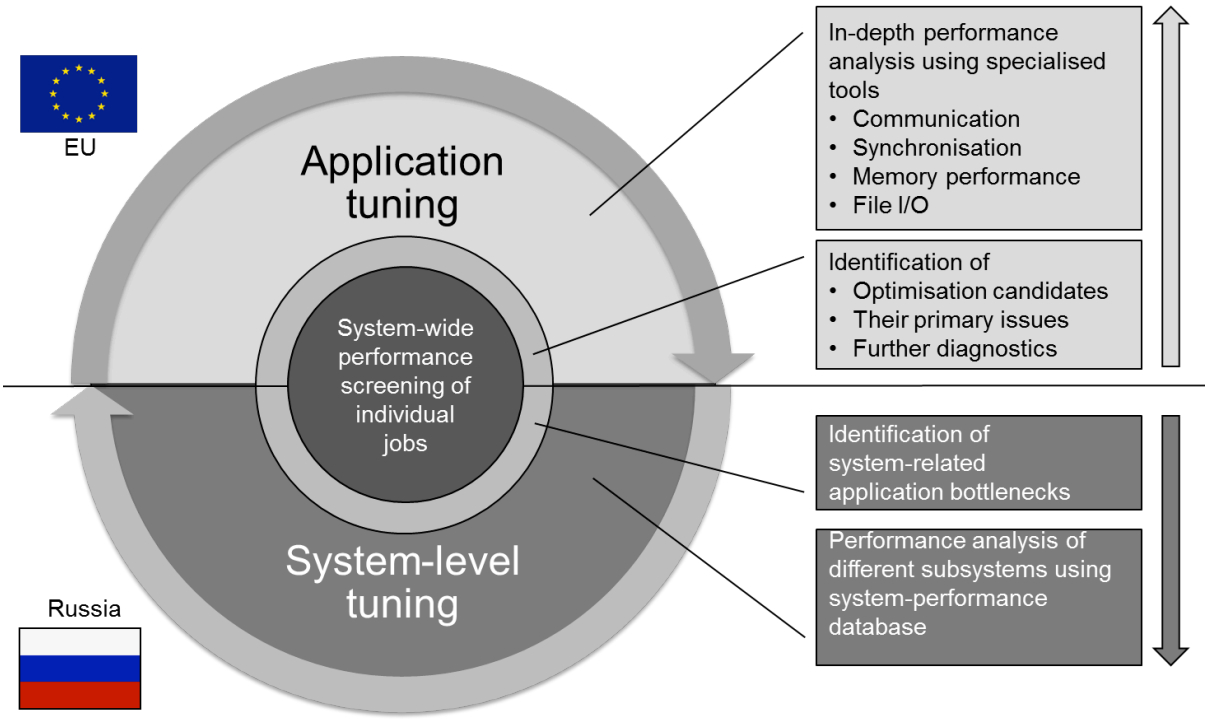

Figure 1: System-level tuning (bottom), application-level tuning (top), and system-wide performance screening (centre) use common interfaces for exchanging performance properties.

The HOPSA project (HOlistic Performance System Analysis) therefore sets out for the first time to develop an integrated diagnostic infrastructure for combined application and system tuning. Using more powerful diagnostic tools application developers and system administrators will easier identify the root causes of their respective bottlenecks. With the HOPSA infrastructure, it is more effective to optimise codes running on HPC systems. More efficient codes mean either getting results faster or being able to get higher quality or more results in the same time.

The work in HOPSA is carried out by two coordinated projects funded by the EU under call FP7-ICT-2011-EU-Russia and the Russian Ministry of Education and Science, respectively. Its objective is the

new innovative integration of application tuning with overall system diagnosis and tuning to maximise the scientific output of our HPC infrastructures. While the Russian consortium will focus on the system aspect, the EU consortium will focus on the application aspect.

At the interface between these two facets of our holistic approach, which is illustrated in Figure 1, is the system-wide performance screening of individual jobs, pointing at both inefficiencies of individual applications and system-related performance issues.

2.2 About this document

This document can be considered as a meta-user guide. It does not replace the user guides of individual performance-analysis tools developed in HOPSA but rather connects them as it shows how to use the tools in a complementary way. At the centre of this document is the so-called lightweight measurement module (LWM^2). Being responsible for the first step in the workflow, the system-wide mandatory collection of basic performance data, the module is covered in greater detail. Special emphasis is given to the interpretation of the job digest created with the help of LWM^2. The metrics listed in this compact report indicate whether an application suffers from an inherent performance problem or whether application interference may have been at the root of dissatisfactory behaviour. They also provide a first assessment regarding the nature of a potential performance problem and help to decide on further diagnostic steps using any of the more powerful performance-analysis tools. For each of those tools, a short summary is given with information on the most important questions it can help to answer. Moreover, the document covers Score-P, a common measurement infrastructure shared by some of the tools. The performance data types supported by Score-P form a natural refinement hierarchy that can be followed to track down and represent even complex bottleneck situations at increasing levels of granularity. Finally, a brief excursion on system tuning explains how system providers can leverage the data collected by LWM^2 to identify a suboptimal system configuration or faulty components.

Please note that not all of the functionality described in the report is available at the time of writing. However, everything should be available at the time the project finishes. If not, it is explicitly indicated.

3. Performance-Analysis Workflow

3.1 Overview

The performance-analysis workflow (Figure 2) consists of two basic steps. During the first step, we identify all those applications running on the system that may suffer from inefficiencies. This is done via system-wide job screening supported by a lightweight measurement module (LWM^2) dynamically linked to every executable. The screening output identifies potential problem areas such as communication, memory, or file I/O, and issues recommendations on which diagnostic tools can be used to explore the issue further. Available analysis tools include Paraver/Dimemas [LGP+1996], Scalasca [GWW+2010], ThreadSpotter [BH2004], and Vampir [NWH+1996]. In general, the workflow successively narrows the analysis focus and increases the level of detail at which performance data are collected. At the same time, the measurement configuration is optimised to keep intrusion low and limit the amount of data that needs to be stored. To distinguish between system and application-related performance problems, some of the tools allow also system-level data to be retrieved and displayed. The system administrator, in contrast, has access to global performance data. He can use this data to identify potential system performance bottlenecks and to optimise the system configuration based on current workload needs. In addition, the administrator can identify applications that continuously underperform and proactively offer performance-consulting services. In this way, it becomes possible to reduce the unnecessary waste of expensive system resources.

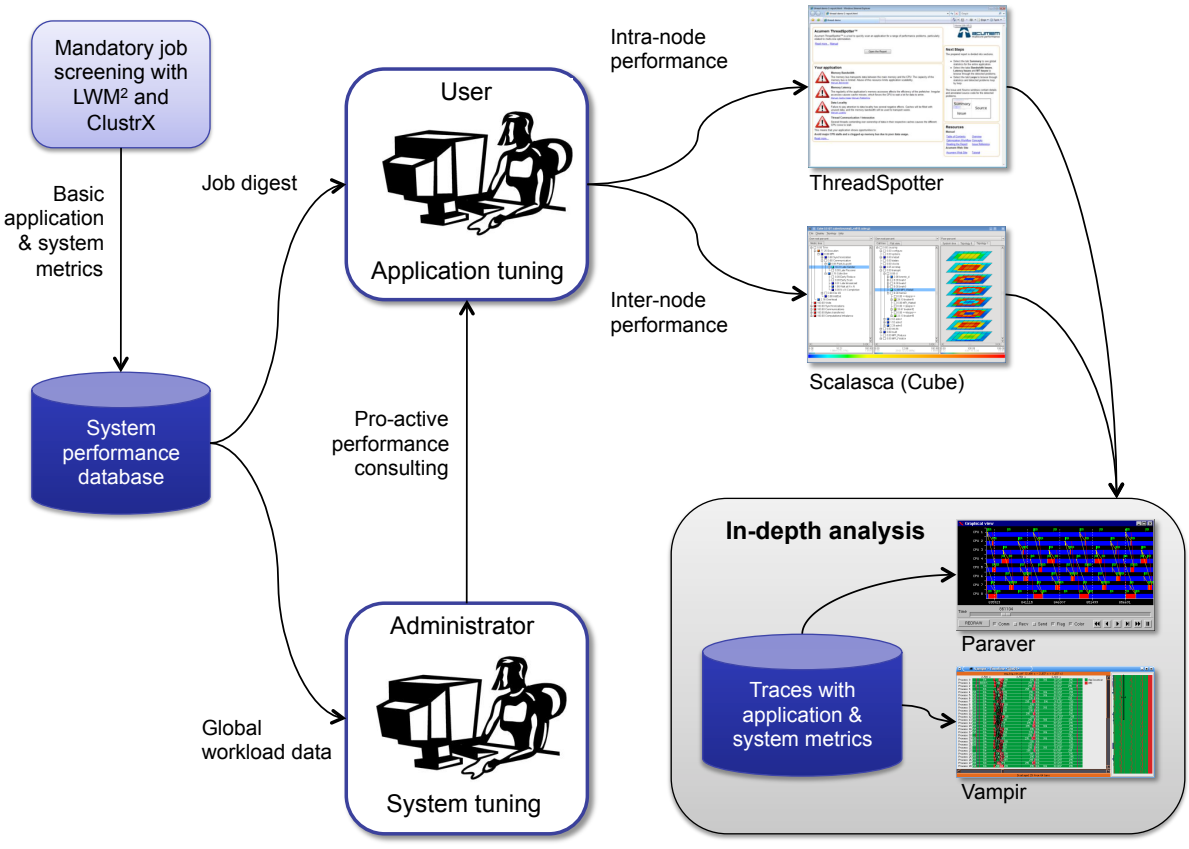

Figure 2: Overview of the performance analysis workflow.

3.2 Performance Screening

This step decides whether an application behaves inefficiently. On the side of the user, nothing has to be done except running the application as usual. Upon application start, LWM^2 is automatically and transparently linked to the executable through library pre-loading. At runtime, the module collects basic performance data via low-overhead sampling. The performance data characterise various aspects such as sequential performance, parallel performance, and file I/O. At the end of execution, the user receives a job digest that contains the most important performance metrics. The digest also recommends further diagnostics in the case certain key metrics show unexpected values, which may often be indicative of a performance problem. If needed, the user can disable LWM^2, for example, to avoid interference with the analysis tools used in subsequent stages of the tuning process.

3.2.1 The Lightweight Measurement Module LWM^2

The lightweight measurement module LWM^2 collects basic performance data for every process of a parallel application. It supports applications based on MPI and multithreaded applications based on POSIX Threads or any higher-level model implemented on top of it, which usually includes OpenMP. Multithreaded MPI applications and applications that additionally use CUDA are supported as well.

To keep the overhead at a minimum, the module applies a combination of sampling and careful direct instrumentation via interposition wrappers. Direct instrumentation is needed to track the state of a thread (e.g., whether it executes inside or outside an MPI function) and to access relevant communication or I/O parameters such as the number of bytes sent or written to disk. Based on the state tracking performed by the instrumentation, sampling partitions the execution time into different components such as computation, communication, or I/O. LWM^2 refrains from direct time measurements as far as possible. Hardware counters deliver basic information on single-node performance. To save storage space, the performance data of individual threads are folded into per-process metrics such as the average number of threads.

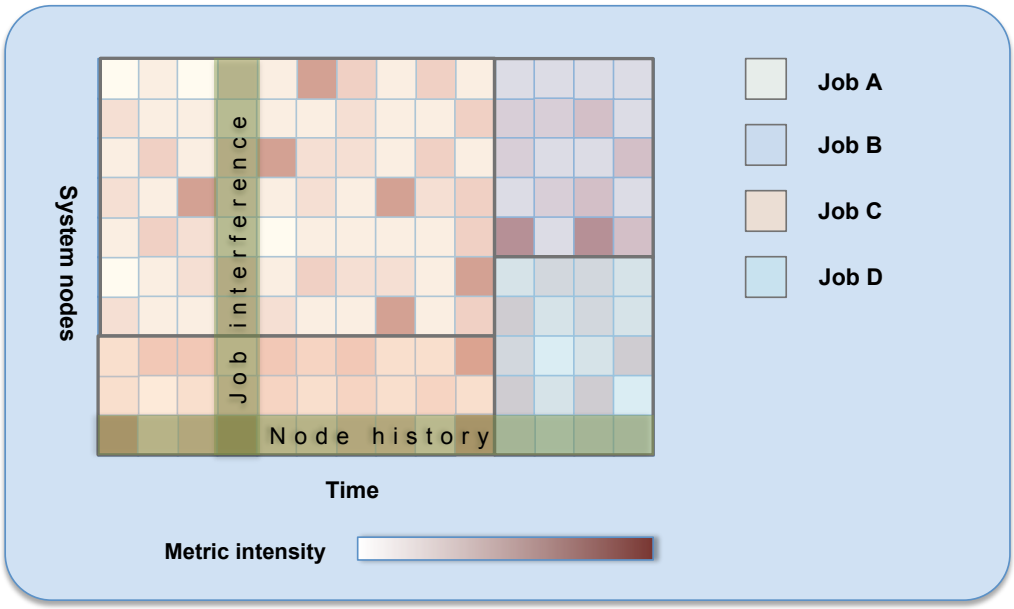

Figure 3: LWM^2 maps selected performance metrics collected during program execution onto a time-and-space grid. The space dimension consists of system nodes, while the time dimension consists of time slices of length 10s, which are synchronized across the entire system to correlate performance data from jobs whose executions overlap in time.

In addition to dividing program execution on the cluster into distinct processes, LWM^2 also divides the time axis into disjoint slices, recording selected metrics related to the use of shared resources at this

finer granularity. The slices have a length of 10s and are synchronized across the entire machine. Together with the location of each process on the cluster, which LWM^2 records along with the performance data, LWM^2 provides performance data for each active cell of a cluster-wide time-space grid (Figure 3). The discretised time axis constitutes the first dimension, the nodes of the system the second one. The purpose of organising the performance data in this way is threefold: First, by comparing the data of different jobs that were active during the same time slice, it becomes possible to see signs of interference between applications. Examples include reduced communication performance due to overall network saturation or low I/O bandwidth due to concurrent I/O requests from other jobs. Second, by looking at the performance data of the same node across a larger number of jobs and comparing it to the performance of other nodes during the same period, anomalies can be detected that would otherwise be hidden when analysing performance data only on a per-job basis. Third, collecting synchronised performance data from all the jobs running on a given system will open the way for new directions in the development of job scheduling algorithms that take the performance characteristics of individual jobs into account. For example, to avoid file-server contention and waiting time that may occur in its wake, it might be wiser not to co-schedule I/O-intensive applications. In this way, overall system utilisation may be further improved.

After the expiration of every time slice, LWM^2 passes the data of the current time slice on to Clustrx Watch, a system-monitoring infrastructure running on each node. Clustrx augments these data with system data collected using various sensors and forwards them to the system performance database, as shown in Figure 4.

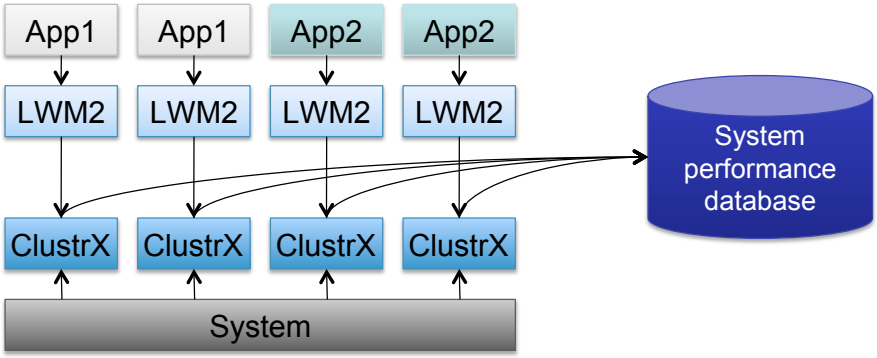

Figure 4: Interaction between LWM^2 and Clustrx Watch.

Finally, studying the performance behaviour of the entire job mix will allow conclusions on the optimal system configuration for the given workload. For example, system providers will learn whether communication requirements were over- or underestimated. Also, performance data of the same application collected over an extended period of time will document the tuning and scaling history of this application.

3.2.2 How to disable LWM^2

While the low runtime overhead of LWM^2 guarantees the undisturbed program execution under normal circumstances, it may interfere with some of the other performance tools that employ similar mechanisms to collect their performance data. For this reason, a user can disable the preloading of LWM^2 in the batch script. The precise way of specifying this option still needs to be determined. The most likely solution is to set an LWM^2-specific batch system variable for disabling LWM^2 for the current batch job.

3.2.3 Job digest

How to access it

The job digest is accessible after job completion through a web-based interface. When the user logs onto the performance database, a list of completed jobs will appear. After selecting a specific job, summary metrics for this particular job will be displayed. For those metrics, for which time-sliced data

is available, the user can view graphs that show the evolution of these metric over time. An important feature is the ability to correlate the evolution of different metrics over time by comparing their graphs. Since the graphs for a single application run cover the same time interval, correlation can be easily observed.

Performance metrics

Below, we provide a detailed list of metrics contained in the job digest. For all metrics where it is applicable, the digest lists minimum, average and maximum values across processes. In addition to defining metrics, we also provide guidance in interpreting them and make recommendations on further analyses if a given metric or group of metrics does not match expectations. In general, it is highly application-dependent whether a metric value should be considered too high or too low. We therefore do not define any fixed thresholds but rather refer to the expectations a user may have.

General information

- Duration of the job in terms of wall clock time
- Number of MPI processes

Message-passing performance

- Time spent in all MPI calls [%]
- Time spent in MPI point-to-point calls [%]
- Time spent in MPI collective calls [%]
- Average size of point-to-point messages [Byte]
- Average size of collective messages sent [Byte]
- Average size of collective messages received [Byte]
- Frequency of MPI point-to-point calls [/s]
- Frequency of MPI collective calls [/s]
- MPI point-to-point transfer rate [Byte/s]. Ratio of the number of bytes sent and the time spent in MPI point-to-point communication
- MPI collective transfer rate [Byte/s]. Ratio of the number of bytes sent and the time spent in MPI collective communication

In general, message passing means communication or synchronisation as opposed to computation and therefore does not directly contribute to the calculation of results. Therefore, communication should be minimized as much as possible and the fraction of time spent in MPI kept low. If the fraction of time spent in MPI calls grows with the number of processes, the application has usually a scalability problem. If communication is dominated by larger numbers of small messages, network latency may be the limiting factor. In contrast, if the majority of messages are large, the limiting factor may be network bandwidth. Asymmetries in the MPI time across processes, indicated by different minimum and maximum times, can be signs of load or communication imbalance, a performance property that usually prevents scaling to larger processor counts. In any case, as the workflow in Figure 2 suggests, Scalasca is the first tool that should be used to analyse communication performance. After identifying the main problems using Scalasca, further, more detailed analysis can follow using Paraver/Dimemas or Vampir. Finally, low communication performance may also be caused by application interference when multiple jobs that run simultaneously compete for the network. This can be verified by comparing the temporal evolution of communication metrics such as the frequency of MPI point-to-point calls with the system-wide communication volume during a given interval.

I/O performance

- Time spent in MPI file I/O calls [%]
- Time spent in POSIX file I/O calls [%]
- Amount of data written to files [Byte]
- Amount of data read from files [Byte]
- Write bandwidth [Byte/s]. Ratio of the number of bytes written to files and the time spent in write functions

- Read bandwidth [Bytes/s]. Ratio of the number of bytes written to files and the time spent in read functions

These metrics indicate whether the application places too much load on the I/O subsystem. The user should always check whether I/O of the given application coincides with I/O of other applications, which is visible in the web-based digest. In such a case, the I/O performance may improve in subsequent runs when such interference is absent. In general, I/O performance is subject to variation and may change significantly between runs. This means, diagnosing an I/O bottleneck usually requires multiple runs under different overall load conditions. Scalasca may help identify expensive I/O calls, while Vampir can help visually discern the overall I/O pattern.

Multithreaded performance

- Average number of threads for the execution: Ratio of the total number of samples and the number of samples taken on the master thread
- Total number of threads in the execution

The average number of threads tells whether the degree of concurrency is as expected. For example, long periods of sequential execution in OpenMP applications may degrade concurrency and limit the benefits of parallel regions for the overall program. Again, Scalasca can help identify places in the code where the concurrency is low, while Paraver and Vampir may provide detailed insights into the change between sequential and parallel phases. Moreover, if the application fails to scale linearly when adding more threads, it could mean that the increased pressure on the memory subsystem causes threads to stall for increasing amounts of time. ThreadSpotter may help figure out the reason.

Sequential performance

- Average cycles per instruction (CPI)
- Fraction of floating-point operations among all instructions [%]
- L1 data cache hit ratio
- Last-level miss frequency

Sequential-performance metrics tell how well the cores of the underlying machine are utilized. If the cycles per instructions are much higher than the theoretical minimum, then memory access latency or pipeline hazards may be the reason. Also, some operations such as complex floating-point operations may simply take longer than others. The fraction of floating-point operations tells to which degree floating-point performance is the dominant theme. A low L1 hit ration usually indicates low locality and may explain a high CPI value. The last-level miss frequency is equivalent to the frequency of main-memory accesses and may point to memory-bandwidth saturation. Note that a platform may miss some of the hardware counters required for the full set of sequential performance metrics or that some of the required hardware counters cannot be measured simultaneously. In this case, LWM^2 provides only a subset of the above metrics. ThreadSpotter is the first candidate to explore memory performance issues. The folding analysis of Paraver can also shed light on high CPI values as it shows correlations with other hardware metrics [SLG+2010].

CUDA performance

- Time spent in CUDA calls [%]
- Average data volume transferred from host to device [Byte]
- Average data volume transferred from device to host [Byte]
- Frequency of data transfers [/s]

These metrics provide just a very rough indicator of CUDA performance. If these metrics show unexpected values, Scalasca may help identify expensive CUDA calls. Paraver and Vampir may give additional insight.

System-oriented metrics

In addition to the more application-oriented metrics listed above, we also plan to include system-oriented metrics related to CPU usage and network communication health, which are collected by Clustrx.

Application interference

As LWM^2 is used with every job running on the system, and the data from the system side is also collected continuously from the complete system, it is possible to present global summary metrics in the job digest. The main examples for application interference are:

- Average I/O load: If the file system was in heavy use by other jobs running on the system at the time the current job was trying to access the file system, it is possible that this interference caused significant performance degradation.
- Average network load: If the communication network was in unusually heavy use at certain time intervals during the execution of the current job, it is important to be able to identify these time periods. If the current job had unusually low network performance in this interval, the reason was probably the interference from other jobs, and not a performance problem with the job itself.

In both of these cases, heavy system load happening at the same time when the current job is trying to use the shared resources leads to performance degradation. Therefore, it is crucial to be able to correlate performance data that was measured at the same time. As these metrics are collected in a time-sliced manner, we can correlate events happening at the same time with a granularity of 10 seconds, which is the default value for the length of time slices.

3.2.4 How to access the performance database

There are multiple use cases for accessing the database, with widely different characteristics:

- When a user accesses the database, he can view metric data collected about his own jobs, which gives him insight into the performance characteristics of a single job, and also allows for comparison between different executions of the same executable.
- When the system administrator accesses the database, he can get a complete overview about all jobs in a given time interval, which allows for pinpointing jobs with sub-standard execution performance characteristics.
- A special case of the previous use case is when a data mining algorithm is ran on the database to pinpoint problematic jobs automatically, making the system administrator's work much more efficient.

3.3 Performance Diagnosis

This step decides why an application behaves inefficiently. It is only needed if the screening identifies a potential performance problem. Depending on the recommendation made by LWM^2, the user chooses one or more of the performance-analysis tools offered by the HOPSA tool environment. The general strategy of the diagnosis is to start with an overview and then to go deeper as more information on the problem's root cause becomes available.

3.3.1 Overview of the performance-analysis tool suite

	Inter-node performance	**Intra-node performance**	**I/O**
Overview	ThreadSpotter	Scalasca (Cube)	Scalsca (Cube)
In-depth analysis	ThreadSpotter, Paraver, Dimemas	Scalasca trace analyser + Cube, Paraver, Vampir	Vampir

Table 1: Classification of tools based on problem class and level of detail.

An overview of the HOPSA performance analysis tool suite is presented in Table 1. For the analysis of intra-node performance, ThreadSpotter is the primary tool, with the possibility of more detailed analyses using Paraver. For investigating inter-node performance, looking at a performance profile using Scalasca's Cube browser is a good starting point. For even more detailed analyses, the results of the Scalasca trace-analyser can be displayed in Cube, or the Vampir and Paraver/Dimemas tools

can be used for a detailed visual exploration of the traces. For understanding I/O-related issues, profiles displayed in the Cube browser give a good overview, while Vampir can be used for more in-depth analysis. A detailed description of individual tools can be found in the appendix.

3.3.2 The Score-P measurement system

The Score-P [MBB+2012] measurement infrastructure is a highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis of HPC applications. It collects performance data that can be analysed using the HOPSA tools Scalasca and Vampir. In addition, it supports the performance tools Persicope [GO2010] and TAU [SM2006] developed outside the HOPSA project. Score-P has been created in the projects SILC and PRIMA funded by the German Ministry of Education and Research and the US Department of Energy, respectively. It will be maintained and further enhanced in a number of follow-up projects including HOPSA.

The main performance data formats produced by Score-P are CUBE-4 [GSS+2012] for profiles and OTF2 [EWG+2011] for event traces. Profiles provide a compact performance overview, while event traces allow the in-depth analysis of parallel performance phenomena. While classic profiles aggregate performance metrics across the entire execution, time-series profiles treat individual iterations of the application's main loop separately, which allows studying the temporal evolution of the performance behaviour. They provide less detail than event traces, but can cover longer executions. Together, the above-mentioned options form a hierarchy of performance data types with increasing level of detail. The main advantage of Score-P is that a user needs to become familiar with only one set of instrumentation commands to produce all theses data types, which can be analysed using the majority of the tools listed in Table 1. Figure 5 provides and overview of the different performance data types supported by Score-P and the tools that can be used to analyse them. Below we cover the individual data types in more detail.

Profiles

Profiles in the CUBE-4 format map a set of performance metrics such as the time spent on some activity or the number of messages sent or received onto pairs of call paths and processes (or threads in multithreaded applications). Metrics with a specialization (i.e., subset) relationship can be arranged and displayed in a hierarchy. The call path dimension forms the natural call-tree hierarchy. Processes and threads are also arranged in an inclusion hierarchy together with hardware components such as the nodes they reside on. In addition, it is possible to define Cartesian process topologies to represent network or virtual topologies. Profiles can be visually explored using the Cube browser. Compared to its predecessor CUBE-3, CUBE-4 files have been optimized for fast writing by storing the metric values in a binary file.

Time-series profiles

Time-series profiles are like normal CUBE-4 profiles except that they maintain a separate sub-tree in the call tree for each iteration of the time-step loop. This allows the user to distinguish individual iterations and to observe the evolution of the performance behaviour along the time axis. Time-series profiles are created by annotating the body of the time-step loop with special instrumentation, which tells Score-P when an iteration ends and when a new one begins. They can be analysed using the normal Cube display. A future version of Cube (to be completed after this project ends) will provide special iteration diagrams that offer an easy way to judge how the performance changes over time. To avoid that profiling data exceeds the available buffer space, the profiling data can be dynamically compressed using an online clustering algorithm. For this purpose, the user specifies the maximum number of iteration clusters the Score-P runtime system should keep in the buffer.

Event traces

Event traces include all events of an application run that are of interest for later examination, together with the time they occurred and a number of event-type-specific attributes. Typical events are entering and leaving of functions or sending and receiving of messages. Event traces produced by Score-P are stored in the Open Trace Format Version 2 (OTF-2), a new trace format whose design is based on the experiences with the two predecessor formats OTF [KBB+06] and EPILOG [WM2004], the former native formats of Vampir and Scalasca, respectively. The main characteristics of OTF-2 are similar to other record-based parallel event trace formats. It contains events and definitions and distributes data

storage over multiple files. In addition, it is more memory efficient, offering the possibility to achieve measurements with less perturbation due to memory buffer flushes. In contrast to OTF, the event traces are stored in a binary format, which reduces the size of the trace files without the need for a separate compression step. OFT-2 traces are the foundation for further analysis. Vampir can display OTF-2 traces visually using different kinds of displays, including a zoomable timeline. The Scalasca trace analyser identifies wait states and their root causes, producing a CUBE-4 file that provides a higher-level view of the application performance data. This is typically recommended to get an idea of key performance issues before visually exploring the traces directly using a trace browser. Moreover, there is on-going work to convert the traces to the Paraver format so that they can be analysed using Paraver (visual exploration) and Dimemas (what-if analysis).

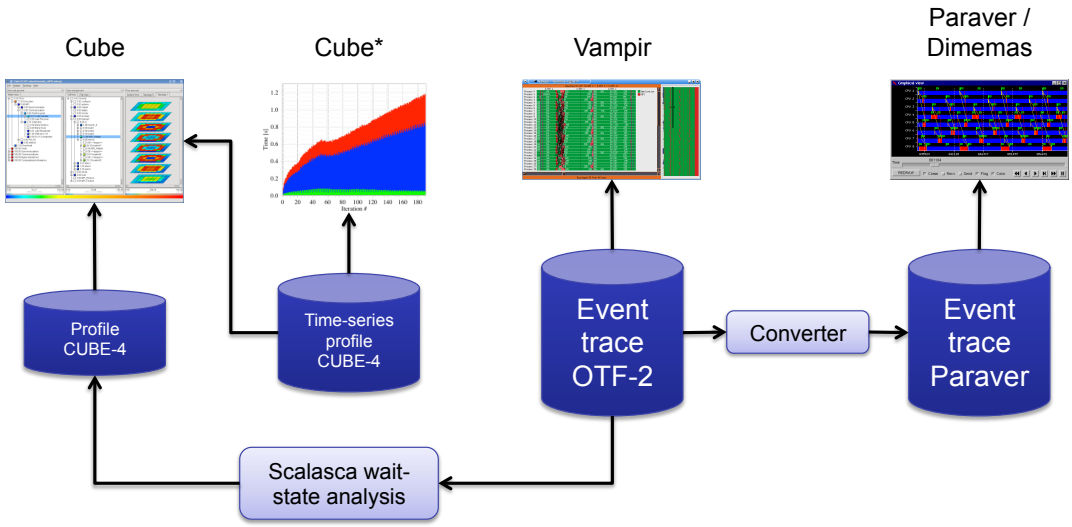

Figure 5: Performance data types supported by Score-P and the tools that can be used to analyse them. The * next to the second mentioning of Cube indicates a display type that will be provided in a future version.

Overhead minimisation

Another important aspect is the quality of the collected performance data in terms of intrusion and their size. To keep both intrusion and data size small, the Score-P measurement system offers a systematic approach of expanding the level of detail while at the same time narrowing the measurement focus:

1. Generate a summary profile with generous instrumentation while measuring the overhead. If the overhead is too large (> 10%), reduce instrumentation, for example, through the application of filter lists. Measure overhead again and iterate until the overhead is satisfactory.
2. Generate a new summary profile with acceptable overhead. This provides an overview of the performance behaviour across the entire execution time and allows the identification of suspicious call paths and processes.
3. Generate a time-series profile, which provides a separate summary profile for every iteration of the time-step loop. This shows to which degree the performance behaviour changes as the simulation progresses and allows the identification of iterations that warrant deeper analysis. A semantic compression algorithm will ensure that the size of time-series profiles stays within reasonable limits.
4. For the identified iterations, generate event traces. Event traces provide the highest level of detail and offer a number of interesting analysis options including automatic wait-state analysis and visual exploration.

3.4 Integration among Performance Analysis Tools

Sharing the common measurement infrastructure Score-P and its data formats and providing conversion utilities if direct sharing is not possible, the performance tools in the HOPSA environment and workflow already make it easier to switch from higher-level analyses provided by tools like Sclasca to more in-depth analyses provided by tools like Paraver or Vampir. To simplify this transition even further, the HOPSA tools are integrated in various ways. With its automatic trace analysis, Scalasca locates call paths affected by wait states caused by load or communication imbalance. However, to find and fix these problems in a user application, it is in some cases necessary to understand the spatial and temporal context leading to the inefficiency, a step naturally supported by trace visualizers like Paraver or Vampir. To make this step easier, the Scalasca analysis remembers the worst instance for each of the performance problems it recognizes. Then, the Cube result browser can launch a trace browser and zoom the timeline into the interval of the trace that corresponds to the worst instance of the recognized performance problems.

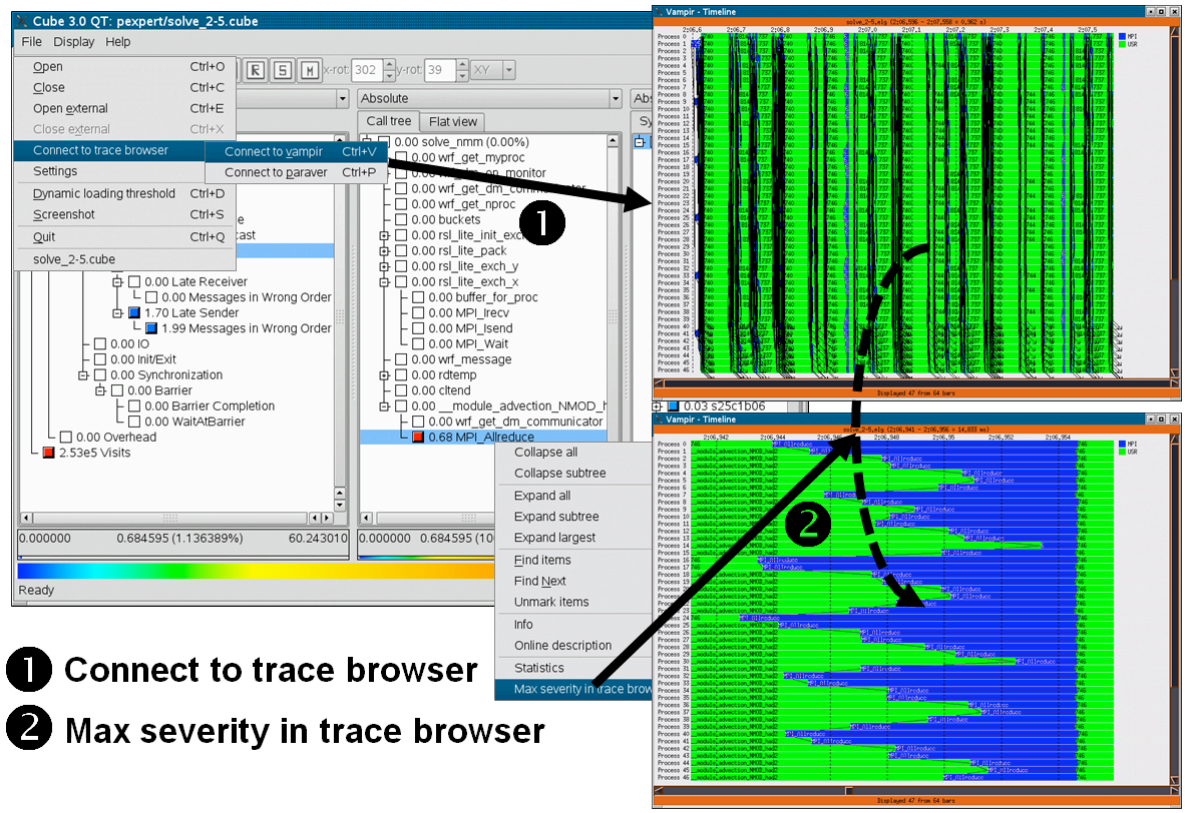

Figure 6. Scalasca →Vampir or Paraver Trace browser integration. In a first step, when the user requests to connect to a trace browser, the selected visualizer is automatically started and the event trace, which was previously the basis of Scalasca's trace analysis, is loaded. Now, in a second step, the user can request a timeline view of the worst instance of each performance bottleneck identified by Scalasca. The trace browser view automatically zooms to the right time interval. Now the user can use the full analysis power of these tools to investigate the context of the identified performance problem.

In the future, the same mechanisms will be available for a more detailed visual exploration of the results of Scalasca's root cause analysis as well as for further analyzing call paths involving user functions that take too much execution time. For the latter, ThreadSpotter will be available to investigate their memory, cache and multi-threading behaviour. If a ThreadSpotter report is available for the same executable and dataset, Cube will allow launching detailed ThreadSpotter views for each call path where data from both tools is available.

3.5 Opportunities for System Tuning

Several opportunities for system tuning arise from the availability of historic performance data collected by LWM^2. First, data on individual system nodes along an extended period of time in comparison to other nodes can be analysed to spot anomalies and detect deficient components. Second, data on the entire workload can be used to improve the understanding of the workload requirements and configure the system accordingly. The insights obtained may guide the evolution of the system and influence future procurement decision. Finally, knowledge of the resource requirements of individual jobs offers the chance to develop resource-aware scheduling algorithms that avoid oversubscription of shared resources such as the file system or the network.

4. Conclusions

The HOPSA project creates an integrated diagnostic infrastructure for combined application and system tuning. Starting from system-wide basic performance screening of individual jobs, an automated workflow routes findings on potential bottlenecks either to application developers or system administrators with recommendations on how to identify their root cause using more powerful diagnostics. This document specifies the performance analysis workflow that connects the different steps. At the same time, it provides an impression of the overall vision behind the project. The high-level description is intended to make it readable also for non-tool experts.

Although the specification is based on long experience with HPC application developers and how they tend to use performance tools, it is a blueprint that needs to be validated in practice. This validation is planned for the last quarter of the project at Moscow State University, once all the components are in place and, in particular, LWM^2 has been fully completed, tested, and integrated into the overall environment. During this validation process, some of the details presented in this document may change and ultimately result in a new revision. We expect though that all major elements will be retained.

Beyond the lifetime of the project, the HOPSA infrastructure is supposed to collect large amounts of valuable data on the performance of individual applications as well as the system workload as a whole. It will be of interest in three ways: to tune individual applications, to tune the system for a given workload, and finally to observe the evolution of this workload over time. The latter will allow the effectiveness of our strategy to be studied. An open research issue to be tackled on the way will be the reliable tracking of individual applications, which may change over time, across jobs based on the collected data. In this way, it will become possible to document the performance history of code projects and demonstrate the effects of our tool environment over time.

5. Bibliography

[GSS+2012] M. Geimer, P. Saviankou, A. Strube, Z. Szebenyi, F. Wolf, B. 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 6–9 2010, volume 7134 of Lecture Notes in Computer Science, pages 463-474, Springer, 2012.

[MBB+2012] D. an Mey, S. Biersdorff, C. Bischof, K. Diethelm, D. Eschweiler, M. Gerndt, A. Knüpfer, D. Lorenz, A. D. Malony, W. E. Nagel, Y. Oleynik, C. Rössel, P. Saviankou, D. Schmidl, S.S. Shende, M. Wagner, B. Wesarg, F. 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.

[EWG+2011] D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. E. Nagel, F. Wolf: Open Trace Format 2 - The Next Generation of Scalable Trace Formats and Support Libraries. In Proc. of the Intl. Conference on Parallel Computing (ParCo), Ghent, Belgium, 2011, volume 22 of Advances in Parallel Computing, pages 481–490, IOS Press, 2012.

[GWW+2010] M. Geimer, F. Wolf, B.J.N. Wylie, E. Ábrahám, D. Becker, B. Mohr: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience, 22(6):702-719, April 2010.

[GO2010] M. Gerndt and M. Ott. Automatic Performance Analysis with Periscope. Concurrency and Computation: Practice and Experience, 22(6):736–748, 2010.

[SLG+2010] H. Servat Gelabert, G. Llort Sanchez, J. Gimenez, and J. Labarta. Detailed performance analysis using coarse grain sampling. In Euro-Par 2009 - Parallel Processing Workshops, Delft, The Netherlands, August 2009, volume 6043 of Lecture Notes in Computer Science, pages 185–198. Springer, 2010.

[SM2006] S. Shende and A. D. Malony. The TAU Parallel Performance System. International Journal of High Performance Computing Applications, 20(2):287–331, 2006. SAGE Publications.

[KBB+2006] A. Knüpfer, R. Brendel, H. Brunst, H. Mix, W. E. Nagel: “Introducing the Open Trace Format (OTF)”, In Vassil N. Alexandrov, Geert Dick van Albada, Peter M. A. Sloot, Jack Dongarra (Eds): Computational Science - ICCS 2006: 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part II, Springer Verlag, ISBN: 3-540-34381-4, pages 526-533, Vol. 3992, 2006

[BH2004] E. Berg, E. Hagersten: StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis. In Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2004), Austin, Texas, USA, March 2004.

[WM2004] F. Wolf, B. Mohr: EPILOG Binary Trace-Data Format. Tech. Rep. FZJ-ZAM-IB-2004-06, Forschungszentrum Jülich (2004)

[LGP+1996] J. Labarta, S. Girona, V. Pillet, T. Cortes, L. Gregoris, DiP: A parallel program development environment, in: Proc. of the 2nd International Euro- Par Conference, Lyon, France, Springer, 1996.

[NWH+1996] W. Nagel, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and Analysis of MPI Resources. Supercomputer, 12(1):69–80, 1996.

6. Appendix

6.1 Descriptions of Individual Tools

6.1.1 Dimemas

Dimemas is a performance prediction tool for message-passing programs. The Dimemas simulator reconstructs the time behaviour of a parallel application using as input an event trace that captures the time resource demands (CPU and network) of a parallel application. The target machine is modeled by a reduced set of key factors influencing the performance that model linear components like the point-to-point transfer time as well as non-linear factors like resources contention or synchronisation. Using a simple model, Dimemas allows performing parametric studies (Figure 7) in a very short time frame. The supported target architecture is a cloud of parallel machines, each one with multiple nodes and multiples CPUs per node allowing the evaluation of a very wide range of alternatives, despite the most common environment is a computing cluster (Figure 8). Dimemas can generate as part of its output a Paraver trace file, enabling the user to conveniently examine the simulated run and understand the application behaviour.

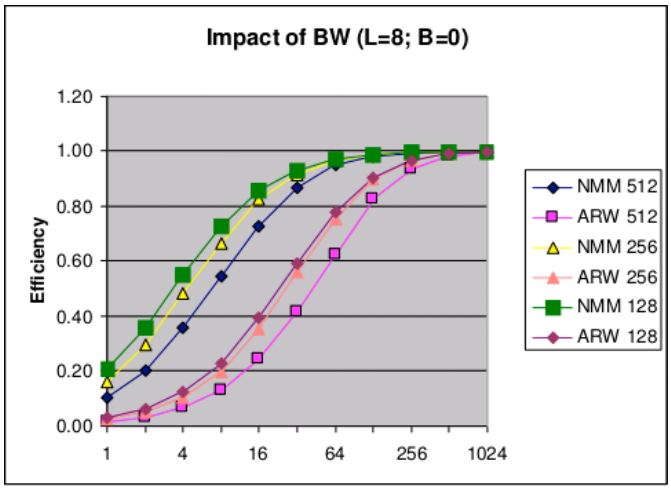

Figure 7: Dimemas parametric study example.

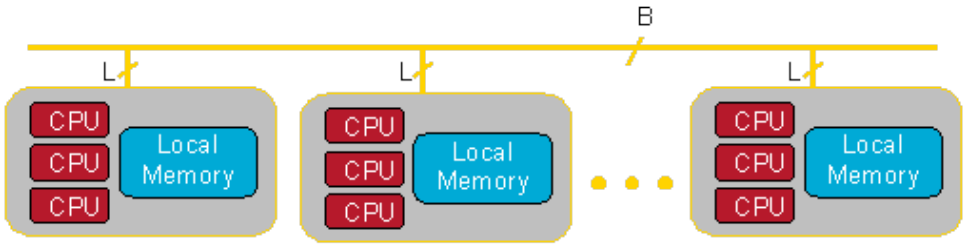

Figure 8: Dimemas's architectural model.

Typical questions Dimemas helps to answer

- How would my application perform in a future system?
- Can increasing the network bandwidth help to improve the application performance?
- Would my application benefit from asynchronous communication?
- Is my application limited by the network or by serialisation and dependency chains in my code?
- What is the sensitivity of my application to different system parameters?
- What would be the impact of optimising or accelerating specific regions of my code?

Supported programming models

Dimemas targets message-passing programming models as well as task-oriented programs. The current instrumentation allows using Dimemas with MPI or MPI+OmpSs applications.

Input sources

The analyses offered by Dimemas rely on event traces in the Dimemas format generated by the Paraver to Dimemas trace translator prv2dim or directly by the runtime measurement system Extrae.

Simulations with Dimemas

Dimemas enables two main types of analyses: **"what-if" studies** to simulate how an application would perform in an hypothetical scenario (e.g., would reducing the network latency by a factor of two have more impact than moving to a CPU twice as fast?), and **parametric studies** running multiple simulations to analyse the sensitivity of the application to the system parameters (e.g., to plot the execution time when varying the network bandwidth from 100Mb/s to 16Gb/s).

A first step to use Dimemas is to translate a Paraver trace file to the Dimemas trace format. It can be the full application execution as well as a representative region with a reduced number of iterations. Then the user specifies through the Dimemas GUI the application trace file to use as input, the architectural parameters of the target machine (such as the latencies and bandwidths for inter-node and intra-node communications, number of network devices...) and the mapping of the tasks onto the different nodes. This information is saved in a file that will be passed as parameter to the simulator. Typically, the user will add an option to generate as output a Paraver trace file that can be later compared with the original run using the Paraver tool.

Instrumentation

Dimemas traces are typically translated from a Paraver trace, but they can also be directly generated by the Extrae tool. Refer to the Paraver section for further details on the available instrumentation mechanisms.

License model

Dimemas is available open source under the terms of the GNU Lesser General Public License (LGPL) v2.1.

Further documentation

- Website: www.bsc.es/dimemas
- Support email: tools@bsc.es

6.1.2 Paraver

Paraver is a very flexible data browser that is part of the CEPBA-Tools toolkit. Its analysis power is based on two main pillars. First, its trace format has no semantics; extending the tool to support new performance data or new programming models requires no changes to the visualiser – just capturing such data in a Paraver trace. The second pillar is that the metrics are not hardwired in the tool but can be programmed. To compute them, the tool offers a large set of time functions, a filter module, and a mechanism to combine two timelines. This approach allows displaying a huge number of metrics with the available data. To capture the expert's knowledge, any view or set of views can be saved as a Paraver configuration file. After that, re-computing the view with new data is as simple as loading the saved file. The tool has been demonstrated to be very useful for performance analysis studies, giving much more details about the application behaviour than most other performance tools.

Performance information in Paraver is presented with two main displays that provide qualitatively different types of information. The *timeline display* represents the behaviour of the application along time and processes, in a way that easily conveys to the user a general understanding of the application behaviour and simple identification of phases and patterns. The *statistics display* provides numerical analysis of the data that can be applied to any user-selected region, helping to draw conclusions on where and how to focus the optimisation effort. See Figures 9 and 10 for an example of Paraver's main displays.

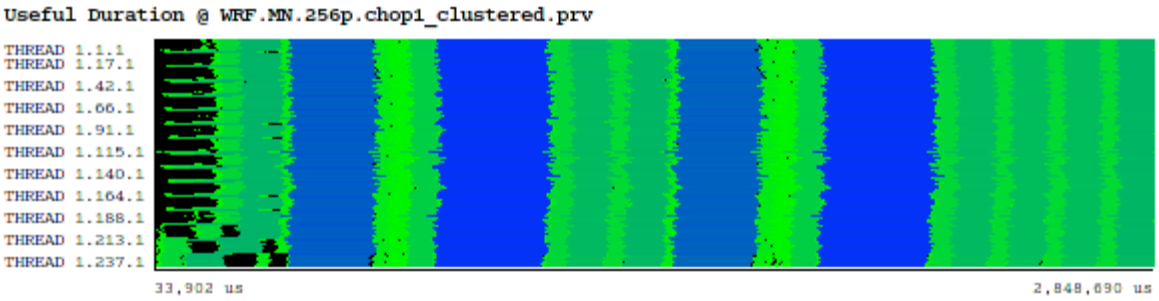

Figure 9: Paraver timeline display.

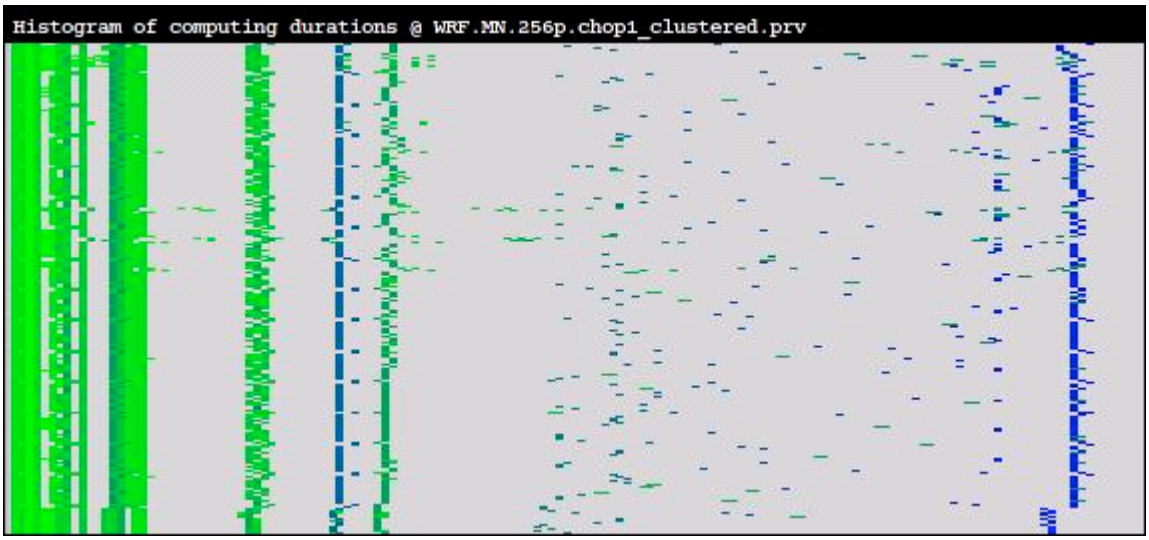

Figure 10: Paraver histogram display.

Typical questions Paraver helps to answer

- What is the parallelisation efficiency and the performance effect of communication?
- What are the differences that can be observed between two different executions?
- Does the behaviour of the application change over time?
- Are performance or workload variations the cause of load imbalances in computation?
- Which performance issues do the microprocessor's hardware counters reflect?

Supported programming models

Paraver is not tied to any programming model as long as the model used can be mapped onto the three levels of parallelism expressed in the Paraver trace. An example of a two-level parallelism would be hybrid MPI + OpenMP applications. The runtime measurement system Extrae that generates Paraver traces currently supports the programming interfaces MPI, OpenMP, pthreads, OmpSs and CUDA.

Input sources

The analyses offered by Paraver rely on event traces in the Paraver format generated by the runtime measurement system Extrae.

Performance analyses

The analysis with Paraver typically starts from a set of pre-conceived configuration files that are available to the user. Each configuration describes a certain view of the performance data, such as the time distribution of functions, MPI primitives or parallel loops called, the value of a given performance metric (e.g., cache misses, floating-point operations, or network bandwidth), and statistics (e.g., profile of the MPI calls – average duration, percentage of time, number of invocations – histogram of the

computation regions duration, correlation between duration and instructions). The tool provides an extensive initial set of configurations that cover those parameters that are usually of highest interest to study, and applying them is as easy as loading a file.

The typical analysis cycle then consists of loading one or more views, zooming into the details of specific processes or code phases, computing histograms and profiles, classifying the data, identifying performance issues, and correlating where these issues happen through the execution, in a process that goes back and forth from the *timelines* to the *statistics.*

The tool offers a very flexible way to combine multiple views, so as to generate new representations of the data and more complex derived metrics. Once a desired view is obtained, it can be stored in a configuration file to apply it again to the same trace or to a different one. Sharing the traces and the corresponding configuration files allows views of the trace and the information obtained to be easily shared.

Instrumentation

Extrae enables four main modes of code instrumentation: manual source-code modification using the Extrae API, library interposition through static linking or dynamic pre-loading, and binary memory image modification at load time using the Dyninst instrumentor. OpenMP constructs are instrumented by wrapping the runtime calls through the dynamic interposition mechanisms, and MPI calls are intercepted through the PMPI profiling interface.

License model

Paraver is available as open source under the terms of the GNU Lesser General Public License (LGPL) v2.1.

Further documentation

- Website: www.bsc.es/paraver
- Support email: tools@bsc.es
- Built-in tutorial (Help →Tutorials)

6.1.3 Scalasca

Scalasca is a free software tool that supports the performance optimisation of parallel programs by measuring and analysing their runtime behaviour. The tool has been specifically designed for use on large-scale systems including IBM Blue Gene and Cray XE, but is also well suited for small- and medium-scale HPC platforms. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronization – and offers guidance in exploring their causes. The user of Scalasca can choose between two different analysis modes: (i) performance overview on the call-path level via profiling and (ii) the analysis of wait-state formation via event tracing. Wait states often occur in the wake of load imbalance and are serious obstacles to achieving satisfactory performance. Performance-analysis results are presented to the user in an interactive explorer called Cube (Figure 11) that allows the investigation of the performance behaviour on different levels of granularity along the dimensions performance problem, call path, and process. The software has been installed at numerous sites in the world and has been successfully used to optimise academic and industrial simulation codes.

Typical questions Scalasca helps to answer

- Which call-paths in my program consume most of the time?
- Why is the time spent in communication or synchronisation higher than expected?
- Does my program suffer from load imbalance and why?

Supported programming models

Scalasca supports applications based on the programming interfaces MPI and OpenMP, including hybrid applications based on a combination of the two. Support for CUDA and StarSs is in progress.

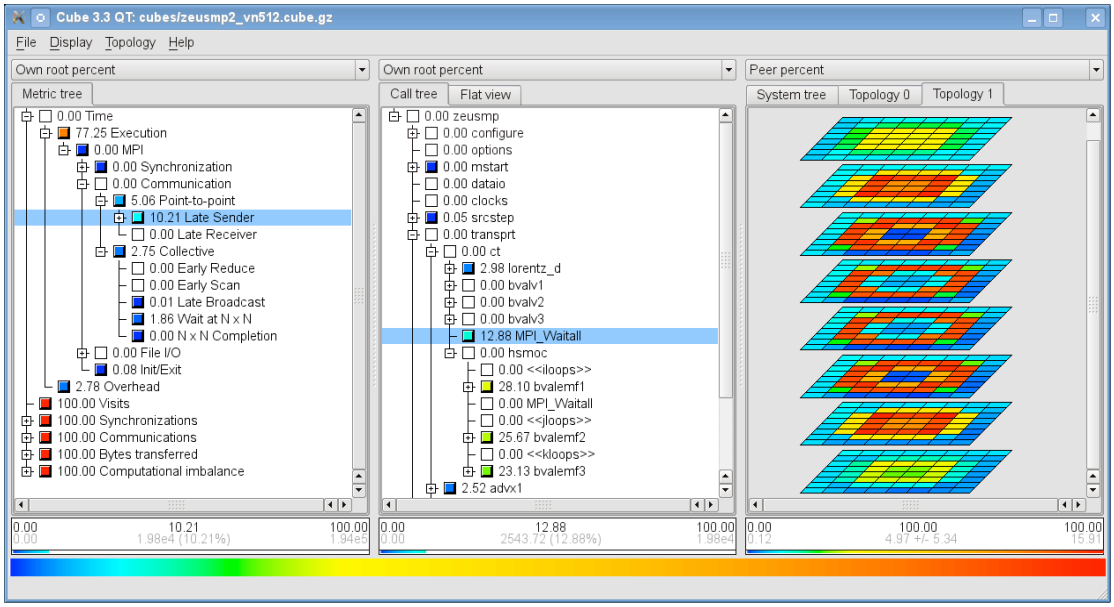

Figure 11: Interactive exploration of performance behaviour in Scalasca along the dimensions performance metric (left), call tree (middle), and process topology (right).

Input sources

The analyses offered by Scalasca rest on profiles in the CUBE-4 format and event traces in the OTF-2 format. Both performance data formats can be generated using Score-P.

Performance analyses

- Summary profile: The summary profile can be used to identify the most resource-intensive call paths or processes. It tells how the execution time and other performance metrics including hardware counters are distributed across the call tree and the set of processes or threads.
- Time-series profile: The time-series profile can be used to analyse how the performance behaviour evolves over time – even if the application runs for a longer period. Essentially, a time-series profile provides a separate summary profile for every iteration of the main loop.
- Wait state analysis: This analysis extracts from event traces the location of wait states. Detected instances are both classified and quantified. High amounts of wait states usually indicate load or communication imbalance.
- Delay analysis: The delay analysis extends the wait-state analysis in that it identifies the root causes of wait states. It traces wait states back to the call paths causing them and determines the amount of waiting time a particular call path is responsible for. It considers both direct wait states and those created via propagation.
- Critical-path analysis: This trace-based analysis determines the effect of imbalance on program runtime. It calculates a set of compact performance indicators that allow users to evaluate load balance, identify performance bottlenecks, and determine the performance impact of load imbalance at first glance. The analysis is applicable to both SPMD and MPMD-style programs.

Instrumentation

User code is instrumented in source code (automatically by compiler or PDT instrumentor, or manually with macros or pragmas). OpenMP constructs are instrumented in source code (automatically by the OPARI2 instrumentation tool). MPI calls are intercepted automatically through library interposition.

License model

The software is available under the New BSD license.

Further documentation

- Website: www.scalasca.org
- Support email: scalasca@fz-juelich.de

- Quick reference guide: installation directory under $SCALASCA_ROOT/doc/manuals/QuickReference.pdf
- Scalasca user guide: installation directory under $SCALASCA_ROOT/doc/manuals/UserGuide.pdf
- CUBE user guide: installation directory under $CUBE_ROOT/doc/manuals/cube3.pdf

6.1.4 ThreadSpotter

ThreadSpotter is a commercial tool that will help programmers optimise their programs with respect to architectural bottlenecks such as cache size and memory system bandwidth and point out inefficient communication modes between threads. Its scope is a single process, including both single-threaded as well as multi-threaded applications.

Some programming styles will exercise the memory system in suboptimal ways that can reduce performance drastically. Examples of these are failure to observe or exploit locality properties in code or data. Inappropriate communication through shared memory between threads may cause the coherence traffic to become a bottleneck.

ThreadSpotter explains the inefficiencies of observed memory access patterns on a high level in a graphical user interface (Figure 12) and provides pointers to suggestions to optimise the code. It offers deep explanations on hardware level to back up the suggestions, educating the user as he uses the tool.

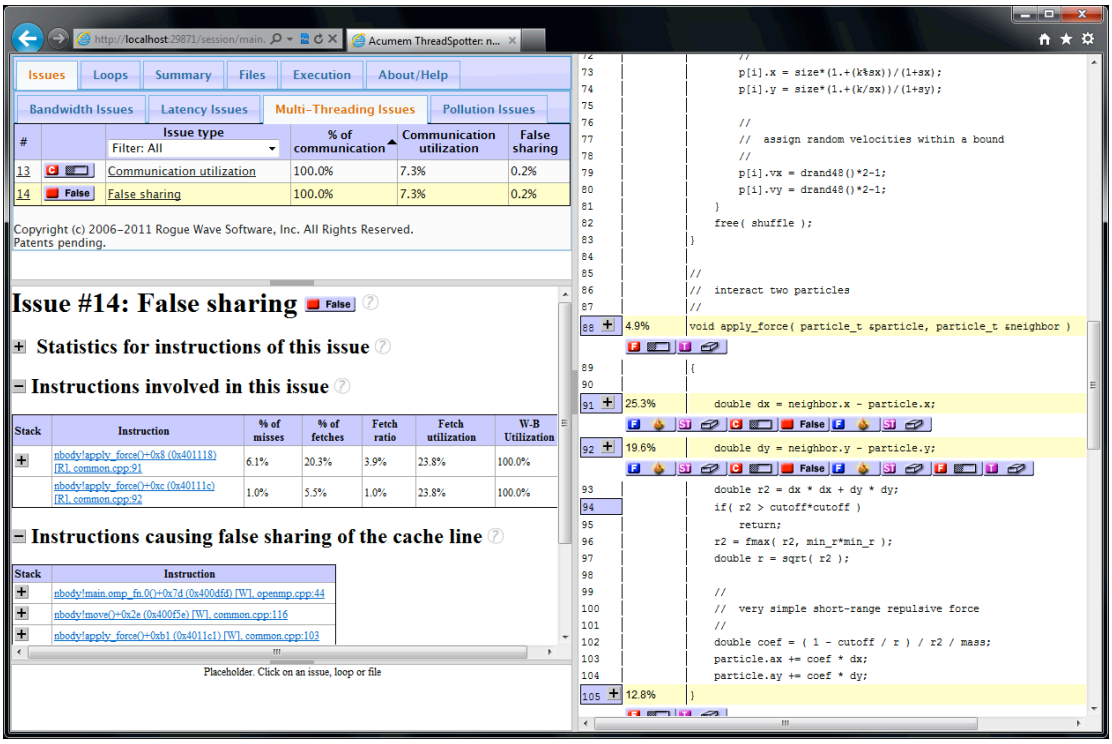

Figure 12: Highlighting a "false sharing" situation. Top left part contains lists of problems. Lower left contains details, and annotated source code is to the right.

Typical questions ThreadSpotter helps to answer

- How does my program abuse the memory system and what can I do about it?
- Do the threads of my program exchange data with each other in an inefficient way?
- When adjusting my program, are the changes actually helping to minimise the footprint of the application?

Supported programming models

ThreadSpotter focuses on a single sequential or multi-threaded process at a time. In distributed environments the user may collect independent information from multiple ranks and investigate the behaviour of these separately. It operates completely on object code level and no particular requirements are expressed on how programs are written, compiled or linked. It supports all threading paradigms.

Input sources

ThreadSpotter includes its own data collection agent, which monitors the application's behaviour as it executes a representative workload. It sparsely collects platform-independent access patterns from the application and stores this data in a small file. This data is then analysed to produce a report. In distributed environments, each rank produces its own data and each one is the source of a separate report, allowing the user to explore runtime behaviour from different parts of the cluster.

Performance analyses

- Overall performance verdict – quickly get a statement on the relative performance and existing problems.
- Summary – graphically and numerically see key metrics from the application as a whole. This can help the user further comparing behaviour between generations of his program on a high level.
- Advice:
  - Spatial locality problems – Explore the high level reasons why the program may not use all of the data that is brought to the cache
  - Temporal locality problems – Find opportunities to reorganise algorithms to use data in the cache more times
  - Prefetch analysis and cache pollution – Instruct the processor to bypass the cache where it makes sense.
  - Multi-threading problems – Find traces of inefficient patterns of data sharing between threads, such as false sharing.
  - Bandwidth and latency: Identify areas in the code where prefetching does not work. Identify program areas taxing the memory bandwidth the hardest.
- Statistics
  - Fundamental cache and bandwidth related metrics: fetch ratio, miss ratio, write-back ratio.
  - Higher level metrics: fetch utilisation, write-back utilisation and communication utilisation.
  - Metrics can be decomposed along different dimensions:
    - Program scope: global, function, loop, instruction
    - Per thread
    - Per type (capacity, coherence, compulsory, …)
- What-if analysis – Perform different experiments from one single fingerprint file:
  - Learn which optimisations are appropriate for different architectures.
  - See what the effect will be of binding threads differently.

Instrumentation

ThreadSpotter uses dynamic binary instrumentation. Thus, it operates on unmodified, production optimised binaries. If debug information is available for the binaries, then ThreadSpotter will be able to point to source code.

License model

The software is available under a commercial license.

Further documentation

- Website: http://www.roguewave.com/products/threadspotter.aspx
- Support email: threadspottersupport@roguewave.com
- Manual: http://www.roguewave.com/support/product-documentation/threadspotter.aspx

6.1.5 Vampir

Vampir is a graphical analysis framework that provides a large set of different chart representations of event-based performance data. These graphical displays, including timelines and statistics, can be used by developers to obtain a better understanding of their parallel program's inner working and to subsequently optimise it. See Figure 13 for an impression of the Vampir GUI.

Vampir is designed to be an intuitive tool, with a GUI that enables developers to quickly display program behavior at any level of detail. Different timeline displays show application activities and communication along a time axis, which can be zoomed and scrolled. Statistical displays provide quantitative results for the currently selected time interval. Powerful zooming and scrolling along the timeline and process/thread axis allows pinpointing the causes of performance problems. All displays have context-sensitive menus, which provide additional information and customisation options. Extensive filtering capabilities for processes, functions, messages or collective operations help to narrow down the information to the interesting spots. Vampir is based on Qt and is available for all major workstation operation systems as well as on most parallel production systems. The parallel version of Vampir, VampirServer, provides fast interactive analysis of ultra large data volumes.

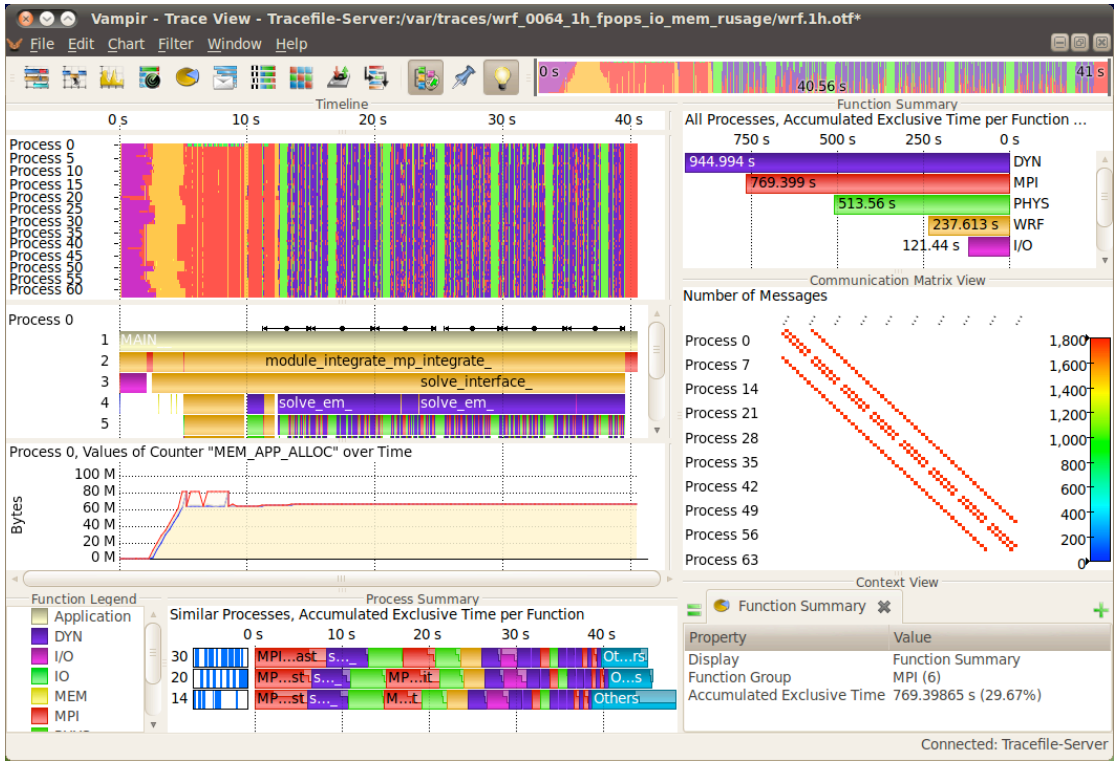

Figure 13: Vampir GUI.

Typical questions Vampir helps to answer

- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

Supported programming models

Vampir supports applications based on the programming interfaces MPI and OpenMP, including hybrid applications based on a combination of the two. Furthermore Vampir also analyses hardware accelerated applications using CUDA and/or OpenCL.

Input sources

The analyses offered by Vampir rest on event traces in the OTF format generated by the runtime measurement system VampirTrace. The next Vampir release also supports OTF2 that is generated by Score-P.

Performance analysis via timeline displays

The timeline displays show the sequence of recorded events on a horizontal time axis that can be zoomed to any level of detail. They allow an in-depth analysis of the dynamic behavior of an application. There are several types of timeline displays.

- Master timeline: This display shows the processes of the parallel program on the vertical axis. Point-to-point messages, global communication, as well as I/O operations are displayed as arrows. This allows for a very detailed analysis of the parallel program flow including communication patterns, load imbalances, and I/O bottlenecks.
- Process timeline: This display focuses on a single process only. Here, the vertical axis shows the sequence of events on their respective call-stack levels, allowing a detailed analysis of function calls.
- Counter data timeline: This chart displays selected performance counters for processes aligned to the master timeline or the process timelines. This is useful to locate anomalies indicating performance problems.

Performance analysis via statistical displays

The statistical displays are provided in addition to the timeline displays. They show summarised information according to the currently selected time interval in the timeline displays. This is the most interesting advantage over pure profiling data because it allows specific statistics to be shown for selected parts of an application, e.g., initialisation or finalisation, or individual iterations without initialisation and finalisation. Different statistical displays provide information about various program aspects, such as execution times of functions or groups, the function call tree, point-to-point messages, as well as I/O events.

Instrumentation

Application code can be instrumented by the compiler or with source-code modification (automatically by the PDT instrumentor, or manually using the VampirTrace/Score-P user API). OpenMP constructs can be instrumented by the OPARI tool using automatic source-to-source instrumentation. MPI calls are intercepted automatically through library interposition.

License model

Vampir is a commercial product distributed by GWT-TUD GmbH. For evaluation, a free demo version is available on the website.

Further documentation

- Website: www.vampir.eu
- Support email: service@vampir.eu
- Vampir manual: installation directory under $VAMPIR_ROOT/doc/vampir-manual.pdf

German Research School
for Simulation Sciences

Published by:
Laboratory for Parallel Programming
German Research School for Simulation Sciences GmbH

September 2012

Schinkelstraße 2a, 52062, Aachen, Germany

Telephone: +49 241 80 99740
Fax: +49 2461 61 1760
Website: www.grs-sim.de