Deliverable D2.3

Tool Validation Report

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

Due date of deliverable: January 1st, 2013

Actual submission date: March 15th, 2013

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

Name of lead contractor for this deliverable: TUD

Abstract: The performance tools of the HOPSA project (ThreadSpotter, Paraver, Dimemas, Scalasca, Vampir) are in daily use on the production systems of the computing centres of the project partners (BSC, JSC, ZIH) or by customers of Rogue Wave where they are applied to application codes of computing centre users. Besides helping the application owners to understand and improve the performance of their codes, it also allows the HOPSA tool developers to test and validate whether the proposed enhancements of work package 2 (described by Tasks T2.1 to T2.3) are working and are useful in the analysis of real-world applications. Some of these use cases are documented in this report. In addition, we applied the HOPSA tools to a set of benchmark codes (SPEC MPI) and large scale application suites (DEEP, Mont-Blanc, Cresta, TEXT) from relevant Exascale EU FP7 projects.

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
BSC	Barcelona Supercomputing Center, Spain
CEPBA	European Center for Parallelism of Barcelona (UPC, BSC)
Cube	Performance report explorer for Scalasca (JSC) and Score-P (GRS, JSC, TUD)
CUDA	Compute Unified Device Architecture (Proprietary Programming Interface for Nvidia GPGPUs)
Dimemas	Message passing performance analysis and prediction tool (BSC)
Extrae	Instrumentation and measurement component for Paraver visualizer (BSC)
GRS	German Research School for Simulation Sciences GmbH, Aachen, Germany
GPGPU	General Purpose Graphical Processing Unit
GUI	Graphical User Interface
HMPP	Hybrid Multicore Parallel Programming (Proprietary Programming Model for Heterogeneous Architectures)
HOPSA	HOlistic Performance System Analysis. EU FP7 project
HPC	High Performance Computing
H4H	Hybrid Programming For Heterogeneous Architectures. EU ITEA2 project
I/O	Input/Output
JSC	Jülich Supercomputing Centre (of Forschungszentrum Jülich GmbH), Germany
MPI	Message Passing Interface (Programming Model for Distributed Memory Systems)
OpenCL	Open Computing Language (Programming interface for heterogeneous platforms consisting of CPUs and other execution units like GPUs)
OpenMP	Open Multi-Processing (Programming Model for Shared Memory Systems)
ORNL	Oak Ridge National Laboratory
OTF2	Open Trace Format Version 2
Paraver	Event trace analysis and visualization tool (BSC)
RW	Rogue Wave Software AB, Sollentuna, Sweden

Scalasca	SCalable Analysis of LArge SCale Applications (Performance instrumentation, measurement and analysis tool from JSC/GRS)
Score-P	Scalable Performance Measurement Infrastructure for Parallel Codes (Community open-source project of GRS, JSC, TUD and others)
SMPSs	Pragma-based programming model for parallel task (Ss = Superscalar) for shared memory parallel computers (SMP) from BSC
SPEC	The Standard Performance Evaluation Corporation
SPEC MPI2007	SPEC's benchmark suite for evaluating performance of compute intensive applications using MPI
UPC	Universitat Politècnica de Catalunya, Barcelona
ThreadSpotter	Commercial memory and multi-threading performance analysis tool (RW)
TUD	Technische Universität Dresden, Germany
Vampir	Visualization and Analysis of MPI Resources (Commercial event trace analysis and visualization tool from ZIH/TUD)
VampirTrace	Instrumentation and measurement component for Vampir visualizer (ZIH/TUD)
ZIH	Zentrum für Informationsdienste und Hochleistungsrechnen. (Center for information services and HPC of TUD).

1. Executive summary

This report documents use cases where the tools which were improved in the course of the HOPSA project (ThreadSpotter, Paraver, Dimemas, Scalasca, Vampir) facilitated the performance analysis of real-world applications. Further enhancements in the efficiency and scalability of the tools are documented.

1.1 The broader context: The HOPSA project

To maximize 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 optimizing 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 for an integration of application and system programming.

The HOPSA project (HOlistic Performance System Analysis) therefore sets out for the first time for combined application and system tuning in the HPC context developing an integrated diagnostic infrastructure. Using more powerful diagnostic tools application developers and system administrators can more easily identify the root causes of their respective bottlenecks. With the HOPSA infrastructure, it is more effective to optimize 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. Its objective is the new innovative integration of application tuning with overall system diagnosis and tuning to maximize the scientific output of our HPC infrastructures. While the Russian consortium focused on the system aspect, the EU consortium focused on the application aspect.

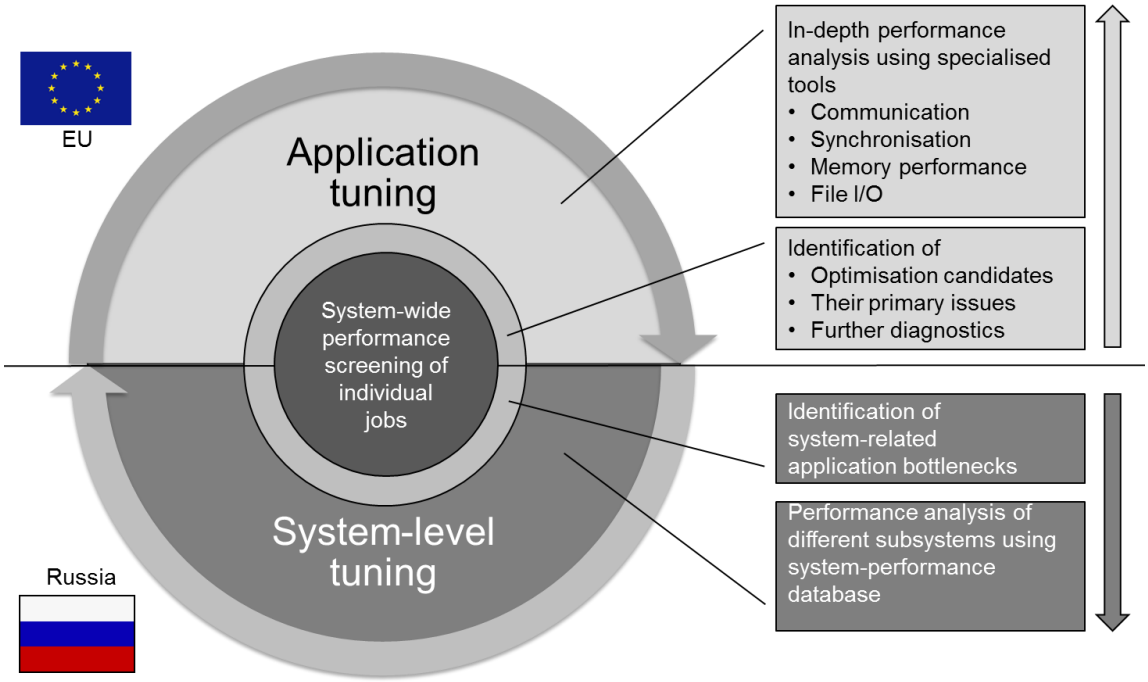

Figure 1: System-level tuning (bottom), application-level tuning (top), and system-wide performance screening (centre) use common interfaces for exchanging performance properties

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

1.2 HOPSA Work package 2: HPC application level analysis

For HPC application tuning, developers can choose from a variety of mature performance-analysis tools developed by the HOPSA-EU consortium: the memory and thread analyzer ThreadSpotter (RW), the trace visualizer Paraver including its measurement system Extrae (BSC), the performance prediction tool Dimemas (BSC), the trace visualizer Vampir (TUD), the performance measurement and analysis tool Scalasca including its result browser Cube (GRS, JSC), and the instrumentation and measurement system Score-P (GRS, JSC, TUD).

Within work package 2 of the HOPSA project, the tools were further integrated and enhanced with respect to scalability, depth of analysis, and support for asynchronous tasking, a node-level paradigm playing an increasingly important role in hybrid programs on emerging hierarchical and heterogeneous systems. The overall objective of work package 2 was to enhance and extend the already existing individual performance measurement and analysis tools of the project partners to make them fit for the analysis of petascale computations and beyond as well as integrating them with each other where useful. The idea here was not to start new research directions but rather to finalize (i.e., “productize”) current research ideas and make them part of the regular tool products. The tools are available as a combination of open-source offerings (Extrae, Paraver, Dimemas, Scalasca, Cube, Score-P) and commercial products (Vampir, ThreadSpotter). At the end of the project (January 2013), a single unified installation package for all tools was provided.

Work package 2 contains all research and technical development which only involves EU partners. Integration with the system-level analysis of the Russian HOPSA partners is the subject of work package 3 and therefore not covered in this deliverable. In the next chapter, the final status of the HOPSA application-level tools is described and the work done as part of work package 2 of the HOPSA-EU project is briefly summarized. A more complete and detailed description of this work is provided by deliverable D1.2 (“Final Progress Report”).

1.2.1 The HOPSA workflow

The indended usage and application of the HOPSA performance tools is specified by the HOPSA performance-analysis workflow (Figure 2). It consists of three basic steps. During the first step (“Performance Screening”), 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 in a second step (“Performance Diagnosis”). If a more simple, profile-oriented aggregated performance overview is not enough to pin-point the problem, a more detailed, trace-based, dynamic performance analysis can be performed in a third step (“In-depth analysis”). Available application performance analysis tools include Paraver/Dimemas [7][8][9], Scalasca [1][2][3], ThreadSpotter [4], and Vampir [5]. The data collected by LWM^2 is also fed into the Clustrx.Watch hierarchical cluster monitoring system [10] which combines it with system and hardware data and forwards it to the LAPTA cluster monitoring and analysis system [11] for further analysis by system administrators.

In general, the workflow successively narrows the analysis focus and increases the level of detail at which performance data is 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, Paraver and Vampir also allow 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 consistently underperform and proactively offer performance-consulting services to the effected users. In this way, it facilitates reducing the unnecessary waste of expensive system resources.

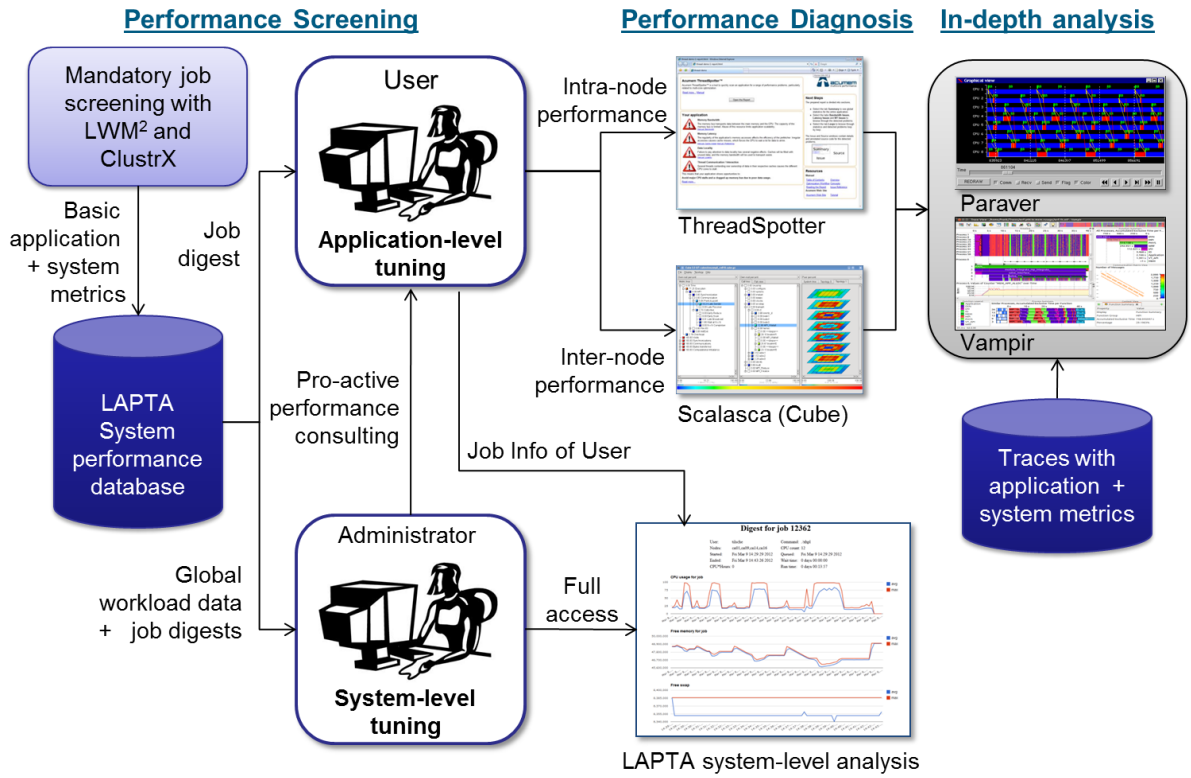

Figure 2: The HOPSA Performance-Analysis Workflow

More details about the HOPSA performance-analysis workflow and tool integration can be found in the Deliverable D3.2 ("Workflow Report") as well in [12]. The light-weight measurement module (LWM^2) is described in more detail in Deliverable D3.3 ("Light-weight Monitoring Module").

1.2.2 HOPSA tool integration

Sharing the common measurement infrastructure Score-P [6] 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 Scalasca 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 (Figure 3). 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. In order to allow the use of Paraver for this analysis, the BSC team implemented an OTF2 to Paraver trace format conversion.

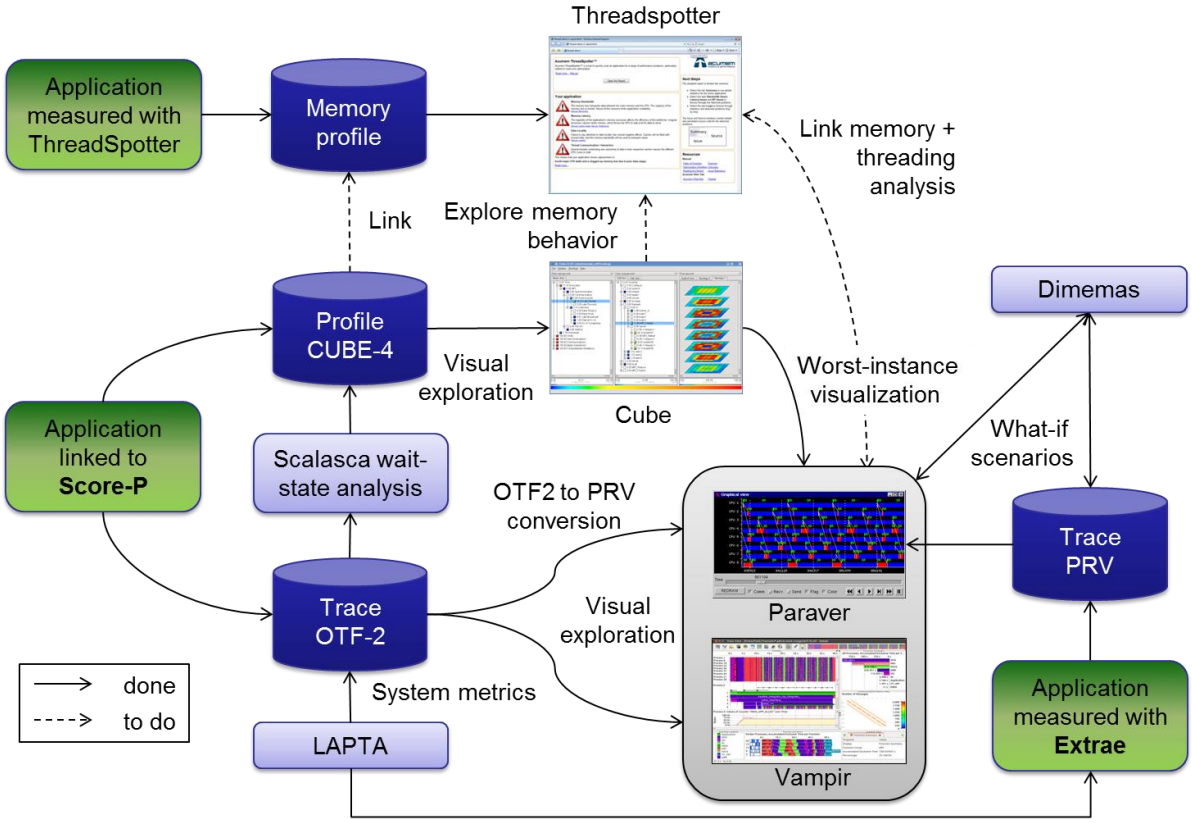

Figure 3: HOPSA Performance Tool Integration

In the future, it is planned to use the same mechanisms 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. The necessary interfaces have been designed and implemented as a prototype during the HOPSA project.

Finally, a tight integration of Dimemas and Paraver allows users to investigate various “what-if scenarios” to further analyze performance properties of their applications.

1.3 Short description of the HOPSA performance tools

This section gives a brief overview about the performance instrumentation, measurement and analysis tools within the HOPSA project. For a longer description see the deliverable D2.2 (“Final Tool Set”). The interaction and integration of the various tools are described by the HOPSA performance-analysis workflow and documented in deliverable D3.2 (“Workflow Report”).

1.3.1 Paraver

Paraver [7][9] is a very flexible data browser that is part of the CEPBA-Tools toolkit developed by BSC and is available for download under an LGPL open-source license. The tool allows a very detailed and powerful exploration of the trace data. Programmable through configuration files, Paraver can visualize performance data via timeline displays (showing metrics per process or thread over time) or histogram displays (showing statistical data). Paraver’s measurement system is Extrae. Extrae is capable of instrumenting applications based on MPI, OpenMP, pthreads, CUDA and StarSs using

different instrumentation approaches. The information gathered by Extrae typically includes timestamped events of runtime calls, performance counters and source code references. Besides, Extrae provides its own API to allow the user to manually instrument his or her application.

1.3.2 Dimemas

Dimemas [8] is a performance analysis tool for message-passing programs. The Dimemas simulator reconstructs the time behaviour of a parallel application on a machine modelled by the key factors influencing the performance. With a simple model Dimemas allows to simulate complete parametric studies in a very short time frame. Dimemas generates as part of its output a Paraver trace file, enabling the user to conveniently examine the simulator run.

1.3.3 Scalasca

Scalasca [1][2][3] supports the performance optimisation of parallel programs by measuring and analysing their runtime behaviour. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronisation – 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 runtime summarisation (aka profiling) and (ii) in-depth study of application behaviour via event tracing. Scalasca, which is jointly developed by JSC and GRS, is available for download under the New BSD open-source license.

1.3.4 Score-P

Score-P [6] is a new open source measurement system developed by TUD and JSC in cooperation with other project partners outside the HOPSA project. Starting 2013, it replaces the measurement systems of Scalasca (EPIK) and Vampir (VampirTrace). It features a new trace (OTF2) and profile (CUBE4) format.

1.3.5 ThreadSpotter

The ThreadSpotter [4] performance optimization technology is developed by Rogue Wave AB – a spin-out from research at Uppsala University in Sweden. While an ordinary binary is running in a production environment, this new performance debugger collects sparse information about its execution behaviour into a ”fingerprint” file. Based on this information, the cache performance of any size cache, any size cache line and several replacement policies can be estimated off-line. ThreadSpotter’s analysis technology also detects performance bugs in the applications, i.e. certain access patterns that result in a sub-optimal performance. ThreadSpotter organizes such performance bugs into four issue groups: bandwidth issues, latency issues, thread interaction issues and cache pollution issues.

1.3.6 Vampir

Vampir (”Visualisation and Analysis of MPI Resources”) is a very well-known event trace visualisation software [5] which is available since 1996 as a commercial product. It offers intuitive parallel event trace visualisation with many displays showing different aspects of the parallel performance behaviour. The corresponding VampirTrace instrumentation and run-time measurement package is available as open source. Vampir is developed by ZIH, TU Dresden and is commercially distributed by the technology transfer company GWT-TUD GmbH.

2. Tool Usage on Benchmarks and Application Suites

In this chapter, we describe the application of the performance tools enhanced during the HOPSA project to a large set of benchmark codes (SPEC MPI) and large-scale application suites (DEEP, Mont-Blanc, Cresta, TEXT) from relevant Exascale EU FP7 projects. It allowed the HOPSA tool developers to test and validate whether the proposed enhancements of work package 2 (described by Tasks T2.1 to T2.3) are working and are useful in the analysis of real-world applications and gave them valuable feedback on the usability and functionality of the tools.

2.1 SPEC MPI benchmark suite

2.1.1 Description of the benchmark codes

The Standard Performance Evaluation Corporation (SPEC) [19] is a non-profit organization that develops, supports, and approves a set of benchmarks for high-performance computers. One of such sets is SPEC MPI2007 [20].

Benchmark	Problem size	Language	Application domain
104.milc	medium	C	Quantum Chromodynamics
107.leslie3d	medium	Fortran	Computational Fluid Dynamics
113.GemsFDTD	medium	Fortran	Computational Electromagnetics
115.fds4	medium	C/Fortran	Computational Fluid Dynamics
121.pop2	medium,large	C/Fortran	Ocean Modeling
122.tachyon	medium,large	C	Parallel Ray Tracing
125.RAxML	large	C	DNA Matching
126.lammps	medium,large	C++	Molecular Dynamics Simulation
127.wrf2	medium	C/Fortran	Weather Prediction
128.GAPgeofem	medium,large	C/Fortran	Heat Transfer using Finite Element Methods
129.tera_tf	medium,large	Fortran	3D Eulerian Hydrodynamics
130.socorro	medium	C/Fortran	Molecular Dynamics using Density-Functional Theory
132.zeusmp2	medium,large	C/Fortran	Physics: Computational Fluid Dynamics
137.lu	medium,large	Fortran	Computational Fluid Dynamics
142.dmilc	large	C	Physics: Quantum Chromodynamics
142.dleslie	large	Fortran	Computational Fluid Dynamics
145.lGemsFDTD	large	Fortran	Computational Electromagnetics
147.l2wrf2	large	C/Fortran	Weather Prediction

Table 1: Description of SPEC MPI2007 benchmarks

SPEC MPI2007 consists of 18 MPI-based independent benchmarks available as source code. It is a benchmark suite developed for evaluation of the performance of compute-intensive MPI-based applications. The primary task of SPEC MPI2007 is to evaluate the performance of CPUs, number of CPUs, MPI Library, communication interconnection, memory architecture, compilers and shared-file system on a given high-performance system.

Programming languages used in this suite vary from case to case. The majority of the benchmarks are written in C or FORTRAN or both. There is only one benchmark implemented in C++ (lammps). Some applications from the suite are able to process different problem sizes (i.e. medium and large). The medium data set is supposed to be used with 4 to 128 processes. The large data set works with 64 to 2048 processes. According to the documentation, average medium benchmarks should run in about 1GB RAM per rank, whereas average large benchmarks should run in about 2GB RAM per rank, without swapping or paging. A detailed list of the SPEC MPI2007 benchmarks and a short description are shown in Table 1 above.

2.1.2 Description of the performed experiments

All experiments were done on JUROPA which is a JSC production system. It consists of 2208 compute nodes. Each node contains two Intel Xeon X5570 quad-core processors and has 24 GB memory.

The main goal of these experiments was to prove that the tools are able to instrument, measure and analyze large MPI-intensive applications. For instrumentation and measurement of the benchmarks from SPEC MPI2007 we used Scalasca 1.4.1, VampirTrace 5.11, and Extrae 2.2.0. For result analysis and visualization, Cube 3.4, Vampir 7.3, and Paraver 4.3. were used. As MPI implementation ParaStation MPI2 was used. All test cases were compiled with the Intel 11.1 compiler.

During the experiments, four problem sizes were used, i.e., mref (the real data set that is required for all medium result reporting), mtrain/ltrain (simple test to prove that executable is functional) and lref (the real data set required for all large result reporting). Almost all test cases were executed with 128 MPI ranks, only a few trace experiments were done with 32 and 16 ranks in order to keep trace file sizes in a reasonable range.

All but one benchmark in SPEC MPI2007 suite could be executed on the system without any modifications. Due to restrictions of the ParaStation MPI2 library used, some of the *MPI_Allgather* and *MPI_Gather* function calls had to be changed in the *fds4* benchmark to use *MPI_IN_PLACE* if the same buffer was passed as the send and the receive buffer.

Instrumentation of all benchmarks with all tools completed successfully without any problems. In order to measure profiles with Scalasca, it was necessary in the case of the *pop2*, *wrf2*, *socorro* and *l2wrf2 benchmarks* to increase the size of internal definition buffers and maximum number of measured call-paths in the measurement system configure setup. Analyzing the profile allows deciding whether filtering is required and to determine the size of per-thread event buffers for tracing measurements. The results from profiling from Scalasca and VampirTrace were similar for all test cases.

In order to keep trace file sizes in an acceptable range, in the most cases filter files were necessary for VampirTrace and Scalasca. For details see Table 2, which shows whether filtering was needed and the sizes of produced profiles and traces. For the *pop2* benchmark filtering alone did not help to complete a successful tracing experiment. In order to record traces for this benchmark, the number of iterations was reduced from 2000 to 1000 in the benchmark input data set. Please note that the sizes of the traces of Scalasca, Vampir, and Paraver cannot easily compared between the three tools, as they record different amounts of information and also the filter mechanisms are quite different.

Benchmark	Scalasca				Vampir			Paraver
	Unfiltered profile	Filter?	Filtered profile	Trace	Unfiltered profile	Filter?	Trace	Trace
104.milc	1.6 MB	yes	1.2 MB	9.3 GB	540 KB	yes	12 GB	1.9 GB
107.leslie3d	203 KB	yes	182 KB	818 MB	528 KB	yes	975 MB	679 MB
113.GemsFDTD	674 KB	yes	579 KB	1.3 GB	548 KB	yes	717 MB	61 MB
115.fds4	760 KB	yes	661 KB	300 MB	548 KB	yes	450 MB	24 MB
121.pop2	1.9 MB	yes	1.7 MB	8.2 GB	548 KB	yes	15 GB	6.3 GB
122.tachyon	772 KB	yes	491 KB	21 MB	536 KB	yes	25 MB	7.3 GB
125.RAxML	2.3 MB	yes	446 KB	474 MB	536 KB	yes	702 MB	514 MB
126.lammps	2.3 MB	yes	1.3 MB	3.3 GB	226 KB	no	22 GB	371 MB
127.wrf2	11.2 MB	yes	7.5 MB	7.2 GB	612 KB	no	57 GB	12 MB
128.GAPgeofem	240 KB	yes	240 KB	9.5 GB	550 KB	yes	1.1 GB	38 MB
129.tera_tf	329 KB	yes	296 KB	516 MB	548 KB	no	78 GB	204 MB
130.socorro	12 MB	yes	1.9 MB	429 MB	620 KB	yes	683 MB	8.4 MB
132.zeusmp2	638 KB	yes	637 KB	154 MB	548 KB	yes	129 MB	92 MB
137.lu	264 KB	yes	254 KB	1.6 GB	454 KB	yes	1.8 GB	798 MB
142.dmilc	1.7 MB	yes	1.2 MB	18 GB	512 KB	yes	22 GB	2.6 GB
142.dleslie	201 KB	yes	180 KB	5.1 GB	556 KB	yes	5.7 GB	8.4 GB
145.lGemsFDTD	676 KB	yes	623 KB	1.9 GB	548 KB	no	14 GB	60 MB
147.l2wrf2	1.3 MB	yes	728 KB	3.4 GB	608 KB	no	15 GB	121 MB

Table 2: Description of profile/trace file sizes and whether filtering was necessary for the different tools

2.1.3 Selected results

For an example, consider the *milc* benchmark. It is an application for quantum chromodynamics. For a first impression of the execution behaviour of the application, the timeline and other graphs provided by the Vampir visualisation browser are useful (see Figure 4).

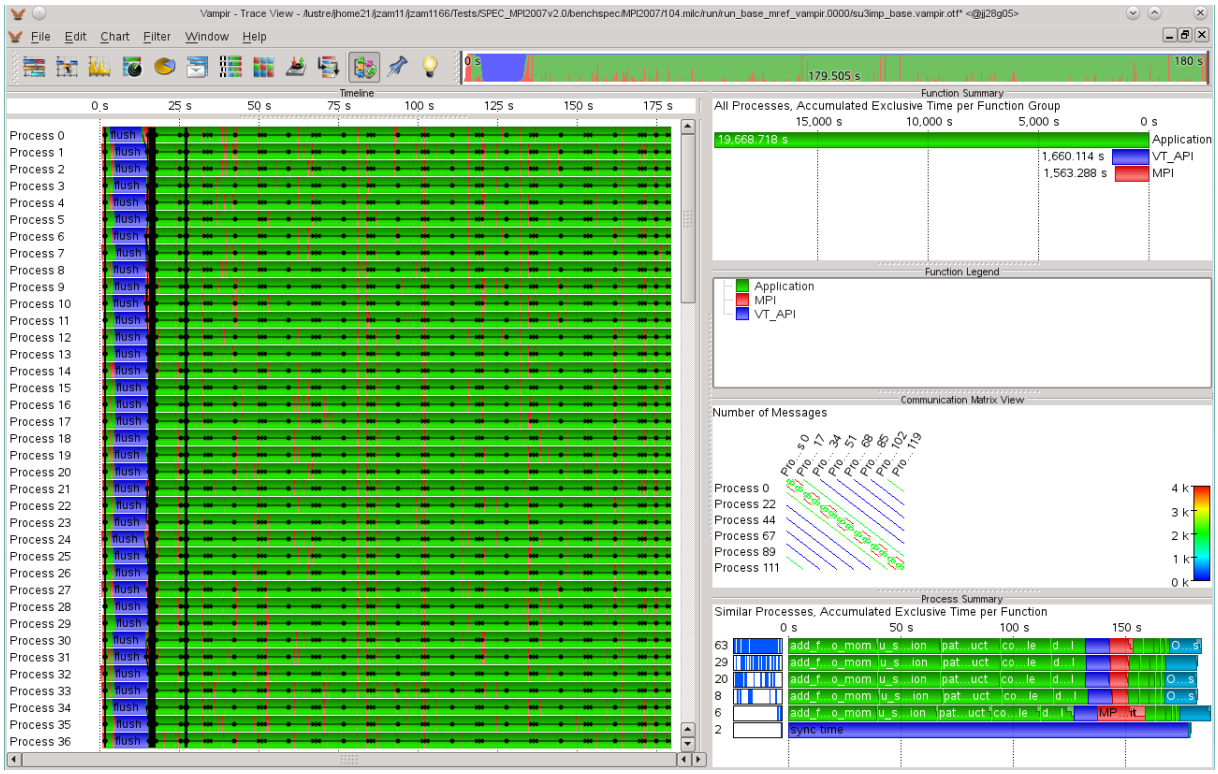

Figure 4: Vampir timeline browser of milc benchmark with 128 processes on JUROPA

From Figure 4 one can notice that the entire execution of the application takes about 5.5 CPU hours (about 3 minutes per process), MPI execution is about 26 CPU minutes (about 12 seconds per process). The communication matrix shows that some processes on the diagonal communicate more than others (red regions in communication matrix). In the process summary graph one can quickly see that two processes are almost idle.

2DP - MPI call profile @ su3imp_base.paraver_.prv <@jj28l07>

	Outside MPI	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Bcast	MPI_Barrier	MPI_Allreduce	MPI_Comm_rank
THREAD 1.1.1	58.84 %	0.25 %	0.51 %	8.44 %	0.00 %	0.00 %	13.05 %	18.89 %
THREAD 1.2.1	77.47 %	0.25 %	0.53 %	1.21 %	0.20 %	0.00 %	1.79 %	18.55 %
THREAD 1.3.1	58.35 %	0.28 %	0.50 %	4.96 %	0.20 %	0.00 %	17.40 %	18.30 %
THREAD 1.4.1	78.15 %	0.28 %	0.51 %	1.38 %	0.20 %	0.00 %	1.28 %	18.20 %
THREAD 1.5.1	65.79 %	0.28 %	0.56 %	10.13 %	0.20 %	0.00 %	4.84 %	18.21 %
THREAD 1.6.1	57.75 %	0.28 %	0.56 %	3.62 %	0.20 %	0.00 %	19.57 %	18.01 %
THREAD 1.7.1	76.78 %	0.30 %	0.60 %	1.42 %	0.20 %	0.00 %	1.88 %	18.82 %
THREAD 1.8.1	63.38 %	0.29 %	0.56 %	5.03 %	0.20 %	0.00 %	12.48 %	18.06 %
THREAD 1.9.1	59.90 %	0.30 %	0.50 %	1.53 %	0.20 %	0.00 %	19.18 %	18.38 %
THREAD 1.10.1	60.34 %	0.30 %	0.54 %	9.11 %	0.20 %	0.00 %	11.01 %	18.51 %
THREAD 1.11.1	62.32 %	0.30 %	0.54 %	3.13 %	0.20 %	0.00 %	15.22 %	18.29 %
THREAD 1.12.1	60.33 %	0.27 %	0.52 %	1.73 %	0.20 %	0.00 %	18.86 %	18.09 %
THREAD 1.13.1	99.80 %	-	-	-	0.20 %	0.00 %	-	0.00 %
THREAD 1.14.1	99.80 %	-	-	-	0.20 %	0.00 %	-	0.00 %
THREAD 1.15.1	99.80 %	-	-	-	0.20 %	0.00 %	-	0.00 %
THREAD 1.16.1	99.80 %	-	-	-	0.20 %	0.00 %	-	0.00 %
Total	1,178.61 %	3.37 %	6.43 %	51.68 %	2.98 %	0.06 %	136.54 %	220.32 %
Average	73.66 %	0.28 %	0.54 %	4.31 %	0.19 %	0.00 %	11.38 %	13.77 %
Maximum	99.80 %	0.30 %	0.60 %	10.13 %	0.20 %	0.00 %	19.57 %	18.89 %
Minimum	57.75 %	0.25 %	0.50 %	1.21 %	0.00 %	0.00 %	1.28 %	0.00 %
StDev	16.44 %	0.02 %	0.03 %	3.13 %	0.05 %	0.00 %	6.87 %	7.95 %
Avg/Max	0.74	0.93	0.89	0.43	0.92	0.84	0.58	0.73

Figure 5: Paraver MPI call statistics of milc benchmark with 16 processes on JUROPA

Additional investigation can be done with the help of statistical information provided by Paraver (see Figure 5). In this case the *milc* benchmark was launched with 16 processes in order to reduce trace sizes. In the figure one can see that most of MPI time was spent in *MPI_Wait* (4.31% of the overall MPI time) and *MPI_Allreduce* (11.38% of the overall MPI time). We can also see the parallel efficiency of the *milc* benchmark, i.e. 74%, in the Avg/Max column of "Outside MPI" which corresponds to the loss of efficiency due to load unbalance.

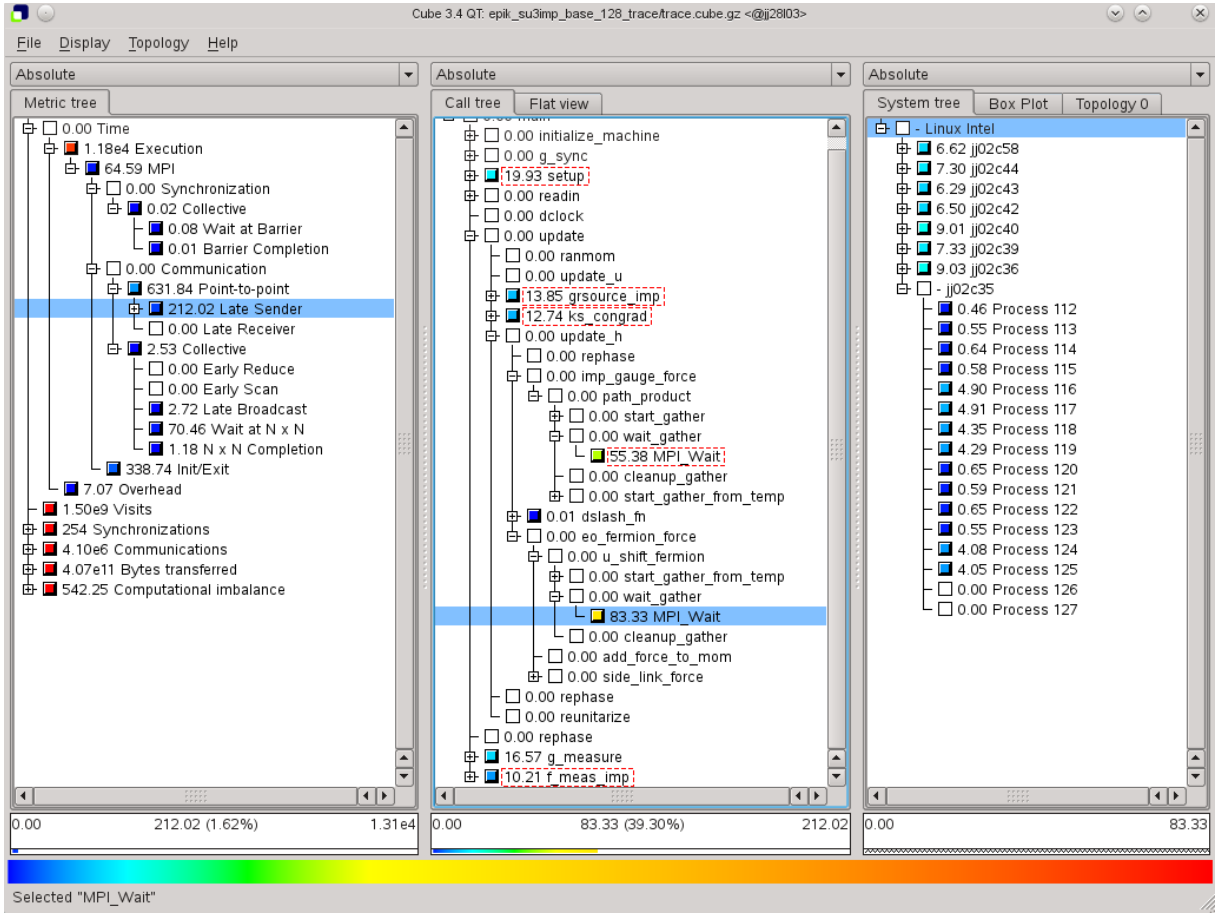

Figure 6: Scalasca trace analysis of milc benchmark with 128 processes on JUROPA

In order to investigate bottlenecks further, let us analyze the *milc* benchmark with the Scalasca trace analyzer. Results are shown in Figure 6. Scalasca's analysis shows that most of the overhead is caused by point-to-point communication and slightly by collective communication. Overhead in this application is mostly caused by waiting time in the form of the "Late Sender" pattern [2] which consumes about one third of the entire point-to-point communication time. Furthermore one can notice that more than half of this waiting time occurs in the *wait_gather* function. In the Cube system tree one can see that the processes in the very last node are unequally utilized. Two processes are idle.

Using a detailed timeline overview of the biggest waiting time caused by "Late Sender", one can investigate the worst instance of this bottleneck in the Vampir visualizer (see Figure 7). One can see easily where the waiting time occurs, i.e. that *MPI_Wait* which completes the reception of messages spawned by *MPI_Irecv* starts much earlier than the matching *MPI_Isend*. In the timeline diagram we can also see that the processors 126 and 127 are not involved in the computation at all, as some processes have much bigger computation time than others. All aforementioned provides hints to the developer for improvement of *milc*'s point-to-point communication implementation.

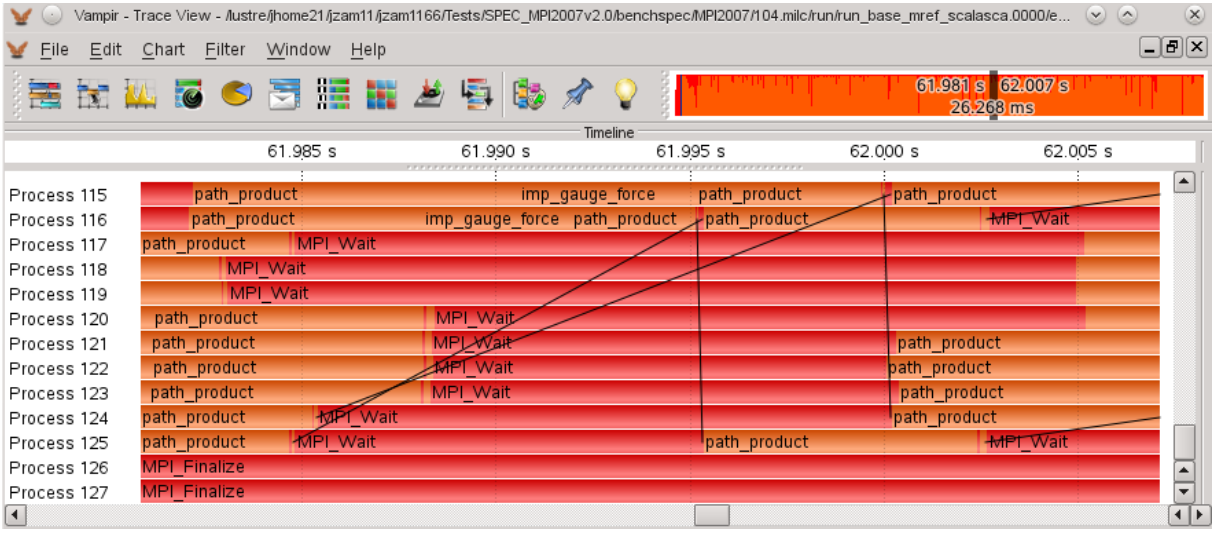

Figure 7: Vampir's timeline visualization of the "Late Sender" worst instance located by Scalasca during execution of milc benchmark with 128 processes on JUROPA

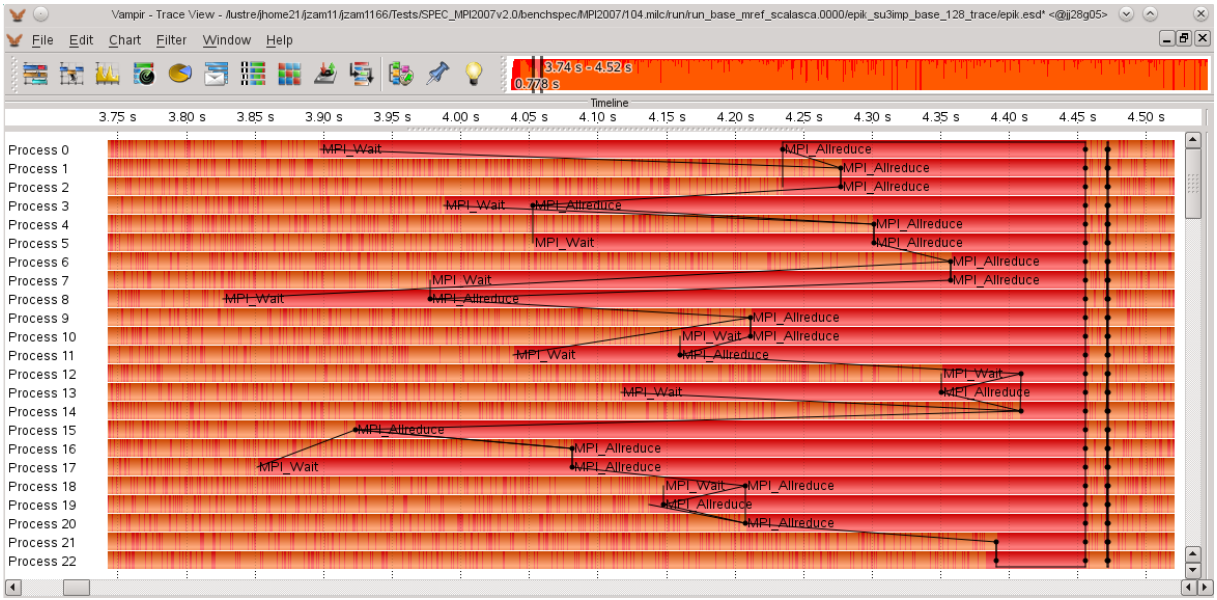

Figure 8: Vampir's timeline visualization of the "Wait at N x N" worst instance located by Scalasca during execution of milc benchmark with 128 processes on JUROPA

The same analysis we can do for collective communication. Here the biggest issue is a "Wait at N x N" bottleneck pattern (5.3% of the overall MPI time). In a "Wait at N x N" pattern, time is spent waiting in front of a synchronizing collective operation until the very last process reaches the operation [2]. We can zoom in to the most severe instance of the "Wait at N x N" pattern in the Vampir browser (see Figure 8). Here one can see that the aforementioned bottleneck is caused by *MPI_Allreduce*.

This example shows how to identify bottlenecks in MPI applications by means of HOPSA performance tools. Integration of these tools helped to find such hotspots much more easily and faster.

2.2 EU FP7 DEEP application suite

HOPSA partners BSC, GRS and JSC are also partners (among others) in the EU FP7 Exascale project DEEP ("Dynamical Exascale Entry Platform") [37]. The main goal is to develop a novel, Exascale-enabling supercomputing platform. To evaluate the DEEP concept and to prove its programmability, selected applications will be ported to the DEEP system. They are used to test the system, to compare its performance with respect to standard architectures, and even to propose improvements to the system's hardware and software. Thus, the applications play a key role in the DEEP project. The simulation codes stem from different fields of science and engineering. The six codes, their application area and their users/developers are:

- NEURON (Brain simulation), EPFL, Switzerland
- iPIC3D (Space weather simulation), KU Leuven, Belgium
- ECHAM/MESSy (Climate simulation), CYI, Cyprus
- AVBP (Computational fluid engineering), CERFACS, France
- TurboRVB (High-temperature superconductivity), CINECA, Italy
- SRMIP (Seismic Imaging), CGGVeritas,France

The application teams use, in cooperation with BSC and JSC tool experts, the Paraver and Scalasca toolsets to support application analysis and performance tuning to facilitate porting and adapting of their application codes to the new innovative DEEP platform. In the first year of the project, Paraver and Scalasca have been both applied to all six application codes. The applications, the experiments performed as well as the results obtained by these experiments are described in detail in the DEEP project deliverable D8.1 ("Applications analysis and code division") [22]. In the following we highlight some of the tool experiments and their outcomes.

2.2.1 Description of the application codes

NEURON

Neuron [15] is a simulation environment for modelling individual neurons and networks of neurons. It provides tools for conveniently building, managing, and using models in a way that is numerically sound and computationally efficient. NEURON is an MPI program that has been used in the Blue Brain Project and ported to highly parallel platforms like the BG/P. NEURON will also be used as starting point for the recently approved Human Brain Flagship project. The main usage of the parallel version has been in a weak scaling mode where networks with more and more neurons are simulated while increasing the number of cores. NEURON also supports a strong scaling model where additional cores are used to accelerate the computation of individual neurons. We will look with Paraver at the behaviour of the parallelization in this strong scaling usage mode. Scalasca has been used to investigate NEURON in the weak scaling mode.

iPIC3D

iPIC3D [16] is a massively parallel Particle In Cell (PIC) Implicit Moment Method (IMM) 3D code for the simulation of plasma behaviour with kinetic scale resolution. The capability to reliably simulate plasma evolution is precious both for the intrinsic scientific relevance of the topic (consider, for example, the multiple challenges still open regarding magnetic reconnection) and as a forecasting tool for space weather related event (i.e., events stirred up by solar activity), which constitute an increasing challenge for modern technology. Space weather related events are many and far reaching; an incomplete list includes damage to spacecraft electronics, GPS signal scintillation, radiation effects on avionics, geomagnetically induced currents in the power system, induced effects on submarine cables and telluric currents in pipelines. iPIC3D is the only example of parallel 3D Implicit Moment Method code and it combines the advantages of using increased time and grid spacing when compared to an explicit code with the benefits given by the parallel implementation. iPIC3D is written in C++, the message passing system used is pure MPI and the only external library used is HDF5 for data output. At each time step, computational particles are moved under the effect of electric and magnetic field defined on a discrete mesh in physical space. Particle information is deposited on the grid through interpolation procedures under the form of moments, i.e. densities, currents and pressures defined on

the mesh, which act as sources for the equation solved for obtaining the fields at the next time step. For the details regarding the algorithm, and in particular for the approximation procedure which allows decoupling the implicitly-written particle and field equation, see [16].

ECAM

At the Cyprus Institute the global atmospheric chemistry climate model EMAC (ECHAM/MESSy Atmospheric Chemistry) is used to simulate atmospheric chemistry and climate change with a focus on the Eastern Mediterranean and the Middle East. The model runs on several platforms including IBM's POWER6 and Intel systems. Climate and atmospheric chemistry simulations are compute-intensive applications and create large data sets. The EMAC model is constituted by a nonlocal meteorological part with low scalability, and local physical/chemical processes with high scalability.

2.2.2 Description of the performed experiments

NEURON

The Paraver analysis of NEURON focused on the strong scaling parallelization, where individual neurons are split over several MPI processes. To understand the issues of such a parallelization, we just focused on the simulation of a single neuron on several cores. In particular, we run the single MPI program within a single 12-core Intel-based node.

The Scalasca experiments focussed on the weak scaling mode of NEURON where one neuron per process is simulated. The runs have been performed on Blue Gene/Q with up to 2048 processes and 32 processes per node, focussing on an architecture-near analysis, but also a trace analysis has been performed.

iPIC3D

The analysis presented of the iPIC3D code is based on a single run of a very short simulation (just five iterations) of a 16 MPI processes run in two nodes of the Judge platform.

ECAM

The Scalasca analysis has been performed on Blizzard, an AIX based Power6 system at DKRZ Hamburg using 128 processes.

2.2.3 Selected results

NEURON

The current strong scaling parallelization within NEURON splits a single cell over several processes. The behaviour of a run of a single neuron with 12 MPI processes is shown in Figure 9. It shows exactly one iteration delimited by the triangular pink region at the end that corresponds to the MPI_allgather calls where different neurons would interchange their interaction values. Within one such iteration several finer timesteps are performed simulating the internal behaviour of a cell. These phases in the timeline show a very important serialization process, where a lot of time is spent by most threads in an MPI_wait call (the dark red areas).

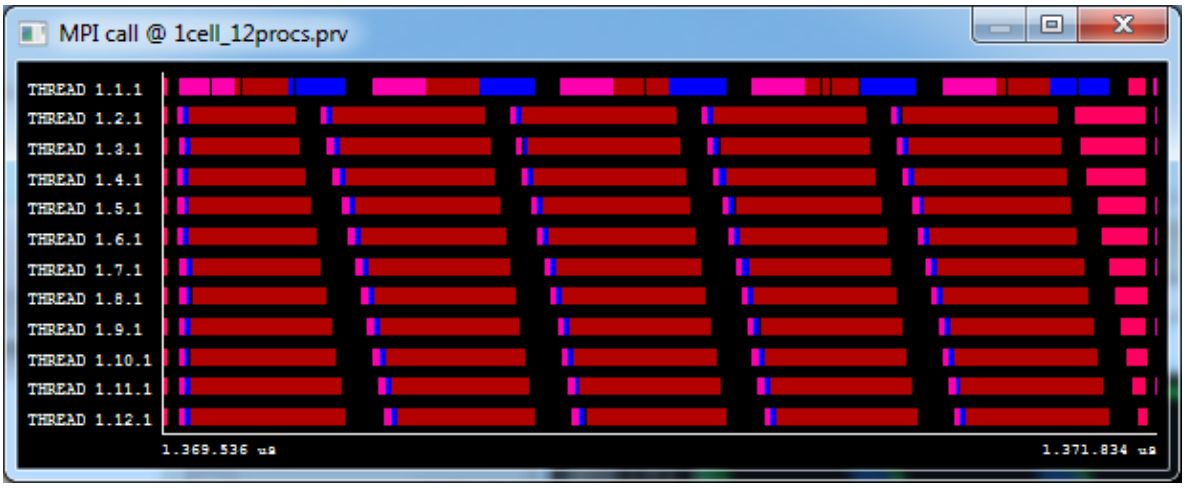

Figure 9: Timeline of MPI call invocations for a single cell simulation in 12 processes

A zoom of the timeline is shown in Figure 10. There is a strong serialization of messages sent from the first process to all others. Every process then performs its local computation and sends data to the first one. The message sizes are different for all processes, but all of them very small (less than 100 bytes).

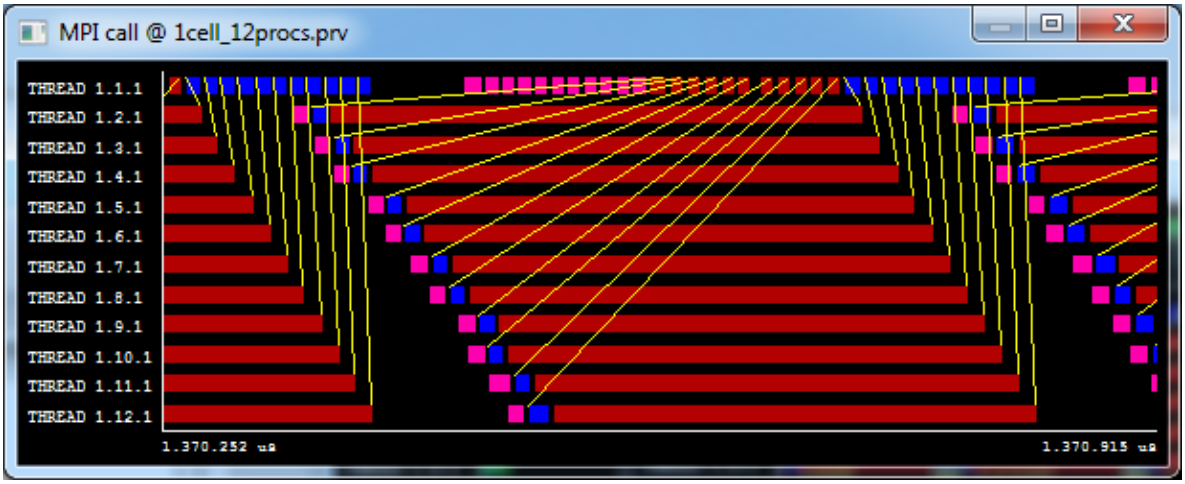

Figure 10: Communication pattern in the internal iteration

The essential serialization behaviour derives from the Schur Complement numerical approach used. Every process computes its local domain and sends data to the first process which then computes the coupling between domains and distributes the result. The computation of the coupling (from last receive and to first send of the first process) is really fast but the MPI overhead of all the MPI calls of this process is the major bottleneck of the application.

The histogram of duration of the computations (Figure 11) also shows some imbalance, where the last processes take more time than the first ones in computing their assigned part of the neuron. The computation of the largest partition of the neuron take in the order of 90 microseconds, which actually means granularity, is very fine.

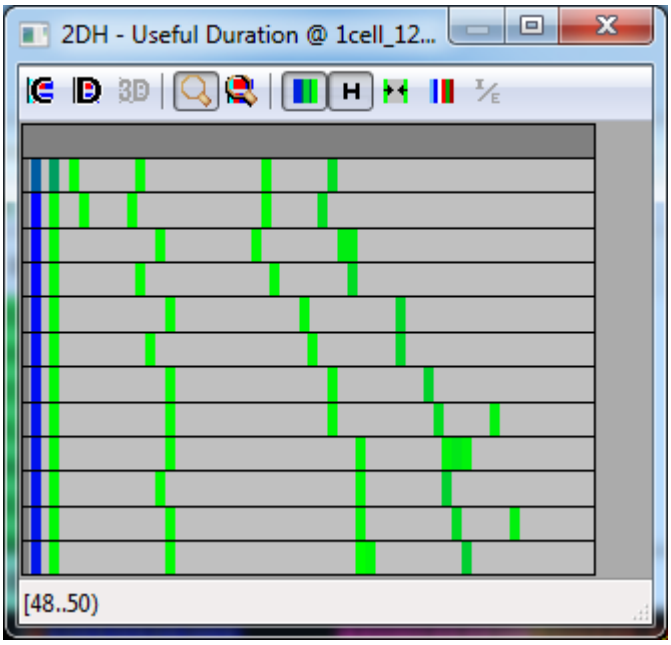

Figure 11: Histogram of duration of computation bursts

The actual load balance of the computation can be quantified with Paraver as a number between 0 and 1. In this case it is 0.48, meaning that just taking into account the computation time outside MPI of the different processes and assuming there were no dependences between them we would lose 52% of the cores because one of them spends significantly more time in computation than the average. Curiously, the bottleneck is process one, which in the histogram seems to compute its domain faster than the other processes. Its excess in execution time comes from the computation of the coupling, but especially from the bookkeeping for all the MPI exchanges. Leveraging the huge flexibility in Paraver we can compute the load imbalance just considering the computation of the local segment of the neuron for each process. This value happens to be 0.85 meaning that we would only lose 15% efficiency due to the decomposition of the neuron.

The analysis clearly suggests using a shared memory programming model to parallelize the computation of one neuron. This would allow a significant reduction in the time to assemble the coupling matrix and might help to better balance the execution.

The Scalasca profile analysis of NEURON focussed on hardware counter measurements to see how NEURON utilizes the Blue Gene/Q hardware and to drive the development of the software to make better use of the hardware features. We used the newly developed derived metrics feature in Cube to analyze metrics on-core (i.e., FLOP/s, CPI), intra-node (i.e., memory bandwidth), and inter-node (i.e., average bytes sent in MPI calls). Figure 12 shows that only very small messages are transferred, 16 bytes in point-to-point communication and 1024 bytes in collective communication.

JSC also looked into the communication behaviour in the considered kernel with a Scalasca trace analysis, see Figure 13. Although only comparatively little time is spend in MPI routines (about 7%), most of it is waiting time, especially waiting time in point-to-point communication with messages arriving in the wrong order (about 3.6% of the total execution time). A more detailed analysis might lead to improvements when going to larger scale of hundreds of thousands of processes.

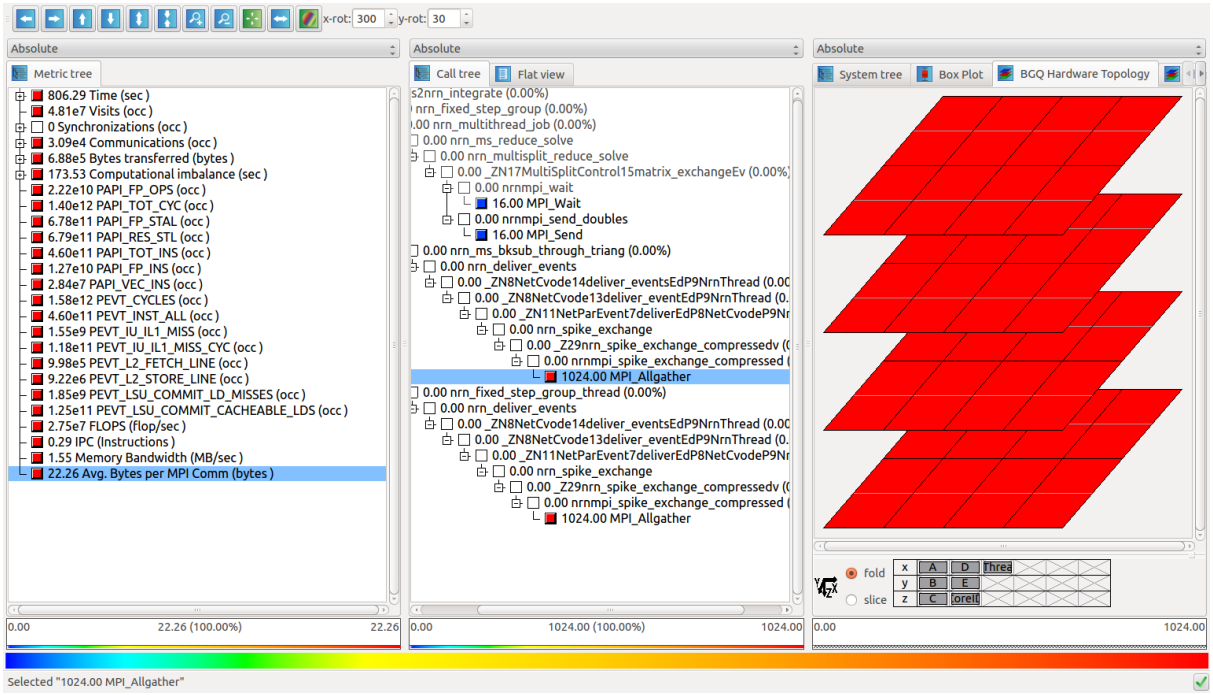

Figure 12: Result of Scalasca's 64-process profile analysis of NEURON shown in Cube. The left panel show the list of measured and derived metrics. Selecting a metric (here: "Avg. Bytes per MPI Comm") displays the distribution of this metric over the call tree of the application (middle panel). Selecting a call site allows investigating the distribution of the metric at this callsite over the machine, displayed as multi-dimensional topology for scalability.

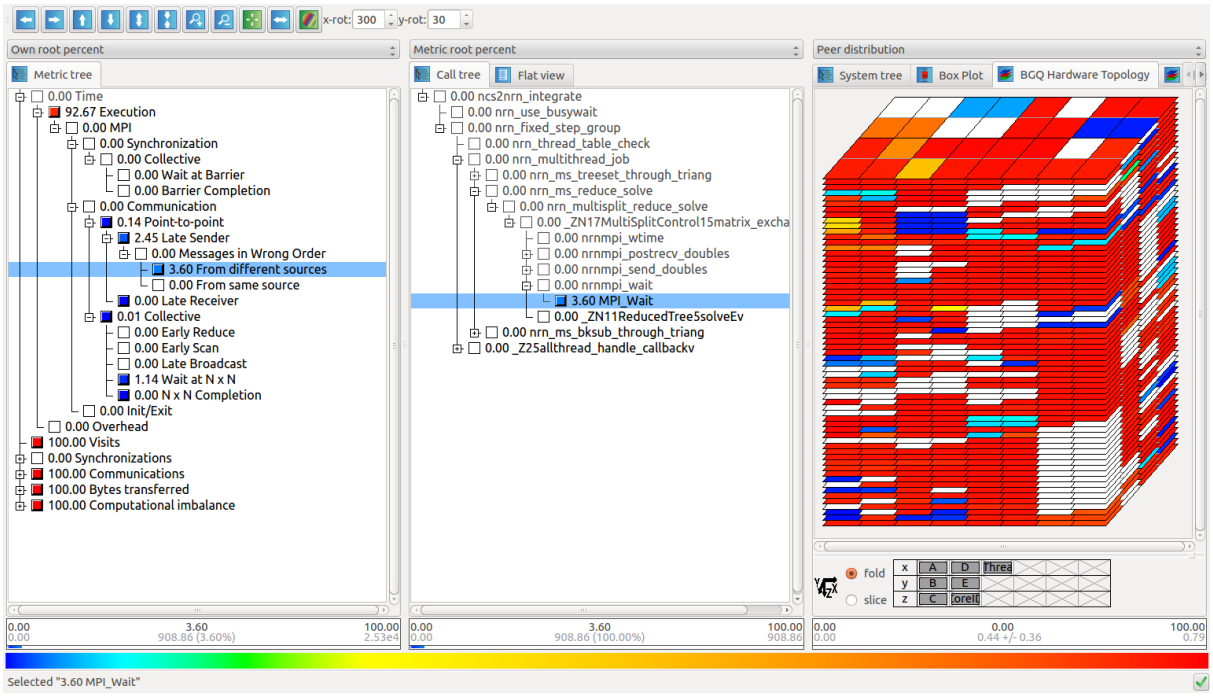

Figure 13: Result of Scalasca's 2048-process trace analysis of NEURON shown in Cube. The figure shows the distribution of the severity of the MPI bottleneck pattern "Late Sender/Messages in Wrong order/From different sources" which Scalasca located in the selected MPI_Wait call.

iPIC3D

From the original trace of iPIC3D we used the Paraver cutting capabilities to chop a time interval of just two iterations to ease handling the analysis. We applied the clustering tool to identify structure in the trace and the result is shown in Figure 14. The green and yellow regions correspond to particle movements while the white, pink and red regions correspond to field solver phases. There are four green and yellow instances in sequence that correspond of the four particle species used in this run. This structural information automatically detected by the clustering tool directly matched the conceptual model in the mind of the application developer who could easily recognize it. The internal structure in the field solver phase shows that the length of the red region increases during the phase and this was a bit of a surprise to the developer at the beginning. She had to look at the detailed routines involved in that region to check that the structure detected by the clustering is really algorithmic. This is a good example of how a tool can detect actual structure in the program that may go beyond the coarse grain conceptual model the developer may have.

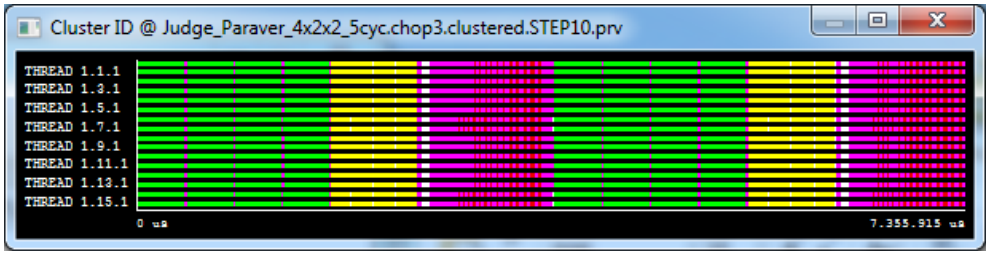

Figure 14: Structure identified by the clustering algorithm

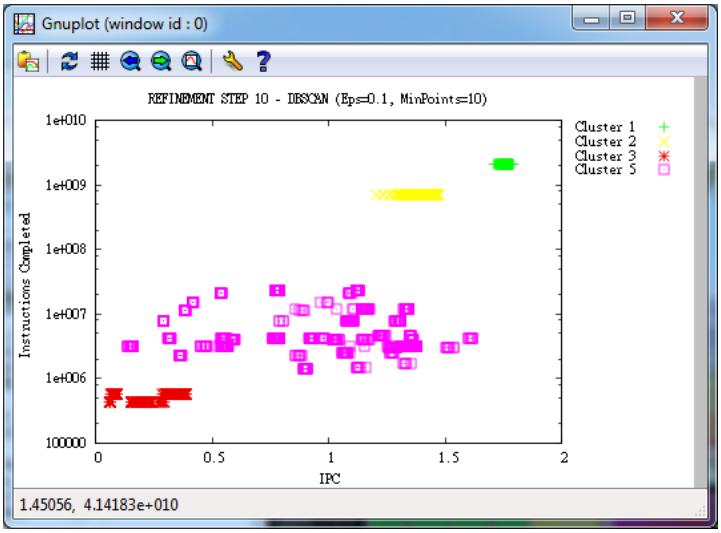

Figure 15: Scatterplot of instructions vs. IPC and clusters identified for the iPIC3D trace

The scatterplot of instructions vs. IPC for those regions is shown in Figure 15. It is interesting to see that the particle regions have a lot of instructions and good IPC (although better for the green region). Given that the yellow region has some variability in IPC we were interested in looking at it in more detail.

Figure 16 show the histogram of IPC and L2 miss ratio (misses/1K instructions), while the timelines for the corresponding metrics are shown in Figure 17. Information the developer was not aware of was the bimodal distribution of those metrics for such region of code. Once she saw it, she was able to rapidly find a physical explanation for such behaviour which she found extremely interesting. The clear bimodal distribution of L2 misses has to do with the type of particles between processes. The two instances of the region with more cache misses happen to correspond to particles with much higher mobility in the physical space.

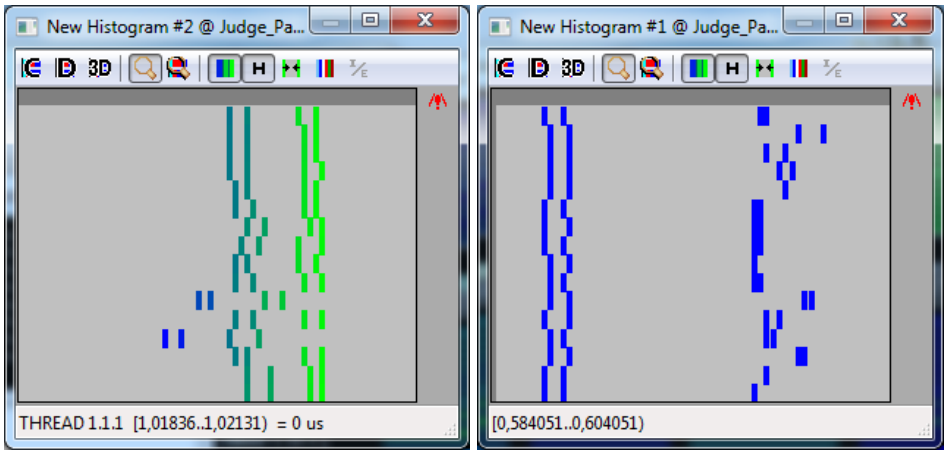

Figure 16: Histogram of IPC (left) and L2 misses (right) for the yellow region

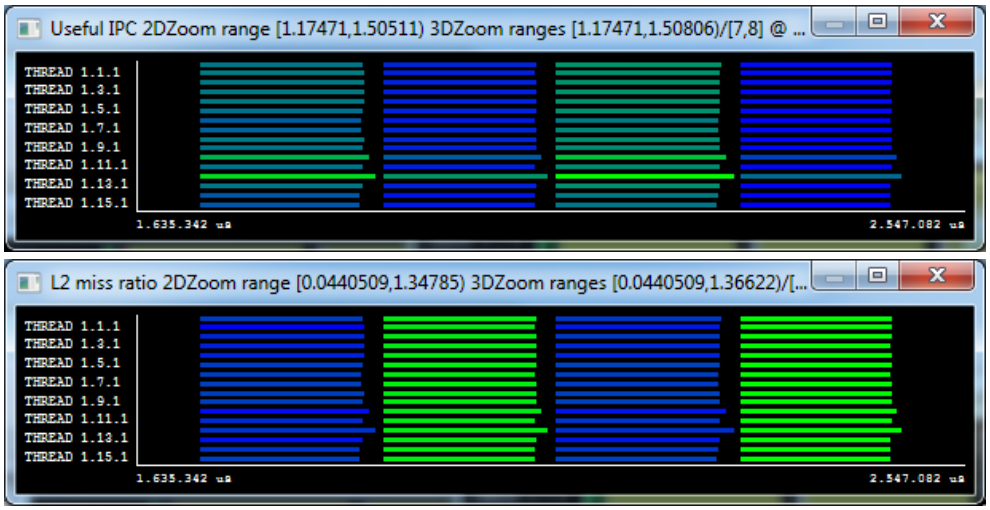

Figure 17: Timeline of IPC (top) and L2 miss ratio (bottom) for the yellow region

ECAM

Using Scalasca we have been able to identify and subsequently remove the *MPI_Broadcast* in the PDEF routine checking and ensuring positive definiteness of the tracer concentrations, acting as a virtual barrier within an inner loop, see Figure 18. Hence, it will be possible to use the shared-memory model for further parallelization. Unfortunately, as can be seen in Figure 19, the load imbalance in this inner loop that causes wait times during global communication is now visible in the following global transpositions GRIDPOINT_TO_FOURIER and FOURIER_TO_LEGENDRE. We need to further investigate how this problem can be resolved.

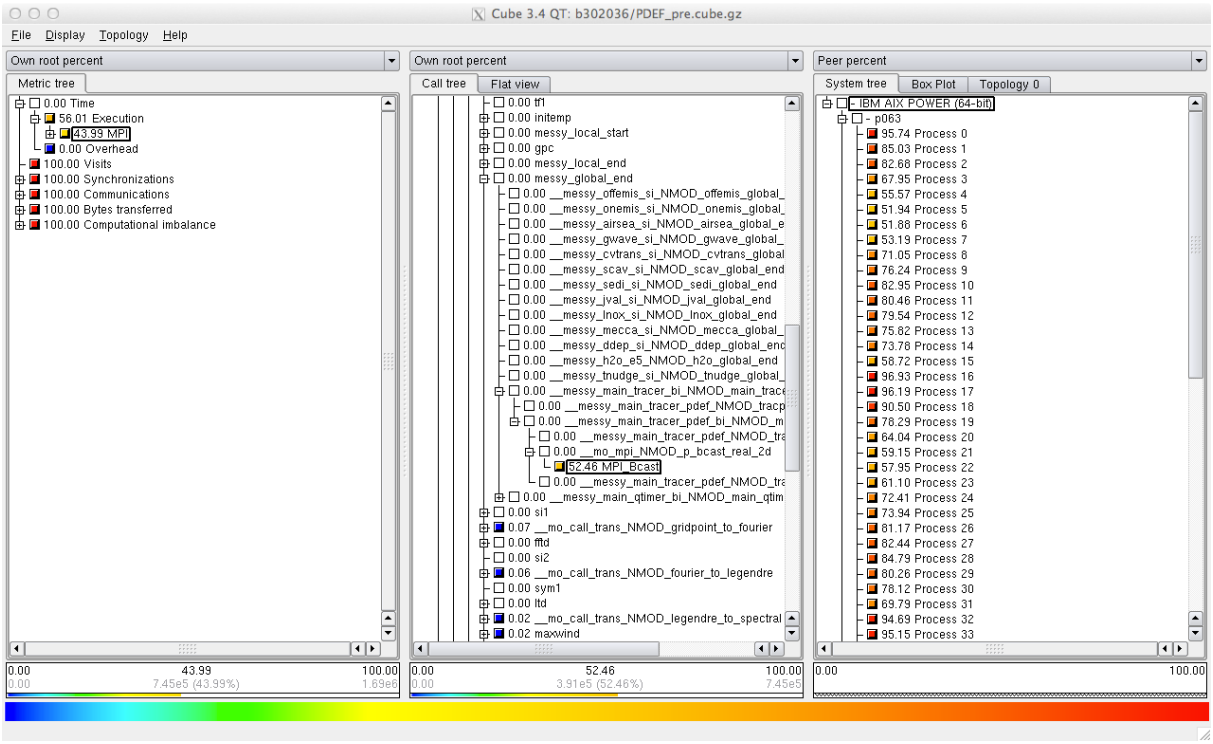

Figure 18: Scalasca analysis of ECAM code showing unexpected high execution time of the selected MPI_Bcast

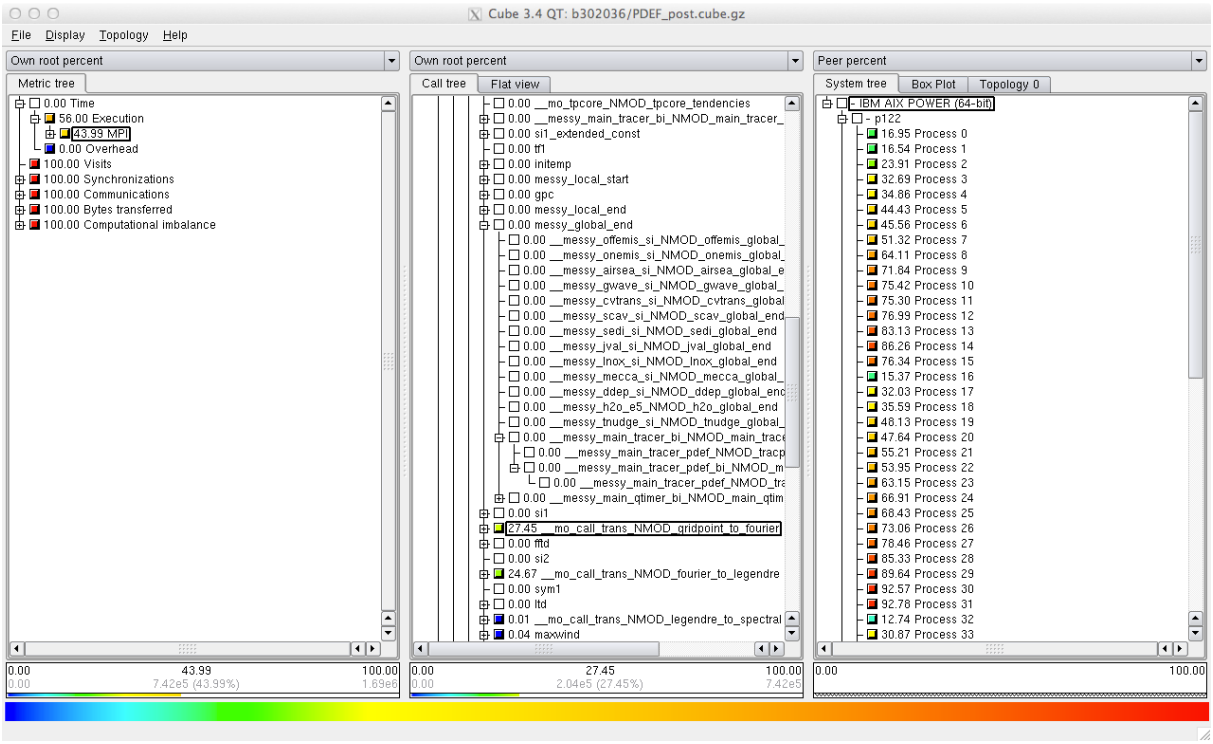

Figure 19: Scalasca analysis of ECAM code after the code optimization. The MPI_Bcast bottleneck is gone but moved into the next phase of the program.

2.3 EU FP7 Mont-Blanc application suite

HOPSA partners BSC and JSC are also partners (among others) in the EU FP7 Exascale project Mont-Blanc [38]. The objective is to design a new type of computer architecture capable of setting future global High Performance Computing (HPC) standards that will deliver Exascale performance while using 15 to 30 time less energy. Like in DEEP, to evaluate the Mont-Blanc concept and to prove its programmability, selected applications will be ported to the Mont-Blanc system. They are used to test the system and system software and to compare its performance with respect to standard architectures. The 11 simulation codes stem from different fields of science:

- YALES2 (Fluid Dynamics)
- EUTERPE (Fluid Dynamics)
- SPECFEM3D (Seismic wave propagation)
- MP2C (Molecular Dynamics), JSC, Germany
- BigDFT (Electornic structure)
- Quantum Expresso (Electronic structure)
- PEPC (Plasma physics), JSC, Germany
- SMMP (Protein Folding)
- ProFASI (Protein Folding), JSC, Germany
- COSMO (Climatology and weather prediction)
- BQCD (Quantum Chromo Dynamics)

The application teams use, in cooperation with BSC tool experts, the Paraver toolset to support application analysis and performance tuning to facilitate porting and adapting of their application codes to the new innovative Mont-Blanc platform. In a later phase of the project, Scalasca will be also available on the Mont-Blanc platform. The applications, the experiments performed as well as the results obtained by these experiments are described in detail in the Mont-Blanc project deliverable D4.1 (“Preliminary report of progress about the porting of the full-scale scientific applications”) [23]. As an example, the Paraver experiments with the SPECFEM3D code is presented in the flowing sections.

2.3.1 Description of the application codes

The software package SPECFEM3D_GLOBE4 [25] is an established petascale-proven production code that simulates three dimensional seismic wave propagation in the time domain based upon the spectral-element method (SEM) on an unstructured mesh of high-order hexahedral finite elements. It is widely used in geophysics (seismology) and in acoustic wave propagation. The SEM is a continuous Galerkin finite-element method with optimised efficiency owing to its tensorised basis functions. In particular, it can accurately handle very distorted mesh elements. SPECFEM3D_GLOBE targets the numerical modelling of seismic wave propagation in the whole Earth, or in large parts of the Earth, following earthquakes. Effects due to lateral variations in compressional-wave speed, shear-wave speed, density, a 3D model of the crust of the Earth, its ellipticity, topography and bathymetry, the presence of the oceans, of rotation, and of self-gravitation are included. Adjoint modelling capabilities and finite-frequency kernel simulations to solve inverse (imaging) problems are also part of the code.

SPECFEM3D_GLOBE is a heavily optimised code that runs in production mode on large and very large scale systems worldwide. It is well tuned for cache locality and also supports GPU-enhanced clusters. Of the three main computational kernels, two perform simple updates on large global vectors of unknowns, which typically contain a few million degrees of freedom per processor core; thus they have the advantage of being trivially parallel, but they have the disadvantage of being memory bound because only a few operations are performed on each degree of freedom accessed in memory. The second computation kernel is more complex, it performs mechanical force calculations and finite-element assembly at each time step based in part on small local matrix–matrix products. The matrices correspond to cutplanes along the three directions of each $(N+1)^3$ spectral finite element described above; they thus have a size of 5 x 5, which is too small to benefit from calling a BLAS3 matrix product library; in SPECFEM3D_GLOBE these products are thus implemented and unrolled manually,

following an implementation introduced by Deville et al. which minimises the number of memory accesses to perform per finite element.

2.3.2 Description of the performed experiments

SPECFEM3D is a long running success story of the analysis capabilities of the BSC tools environment. In the following section we will briefly report on the results of the analyses performed on a former version of the application with Paraver and Dimemas. The executions were done on just a couple hundred cores in the Marenostrum II cluster, a system with 4 PowerPC cores per node and Myrinet interconnect. This example is highly indicative of the capabilities and usage of the tools in order to drive application developments in the most productive direction. The main initial objective of the developers was to determine the best way to overlap communication and computation introducing non-blocking MPI calls. The analysis pointed their development in the direction of improving the domain decomposition technique but specially the locality of the local computation within each process. With the analyses results, developers went back for six months to improve the domain decomposition method, but more important, to improve the numbering scheme within each domain. Only after those optimizations where done, the originally planned usage of non-blocking calls was introduced, giving a marginal advantage compared to the other two optimizations. Now, the MPI version is an example of heavily optimized code that scales to very large numbers of codes.

We will then describe the analyses currently being done on a new version of the code using hybrid MPI+OmpSs parallelization. Our approach is to identify how well is the node level programming model OmpSs able to overlap communication and computation with a simpler source code structure than the current MPI code. OmpSs is being developed at BSC as a very productive programming model with the potential to automatically handle asynchrony and overlap as well as locality and heterogeneity by the NANOS runtime on which it relies. Very detailed understanding of the behaviour of the runtime and the interaction with the application characteristics is needed in order to evolve the OmpSs and NANOS development in the direction that also achieves a performance competitive with heavily optimized MPI codes such as the current SPECFEM3D version.

2.3.3 Selected results

Figure 20 shows the histograms of duration, instructions, IPC and L2 miss ratio for the computation burst between two MPI calls. In each table, there is one row per process and the columns represent the different bins of the corresponding metric. Background grey pixels appear for all those entries in the histogram with no single burst instance for which the metric fall in the corresponding bin. A gradient between light green and dark blue represents the fraction (between low and high) of time spent in computation burst that fall into the corresponding bin.

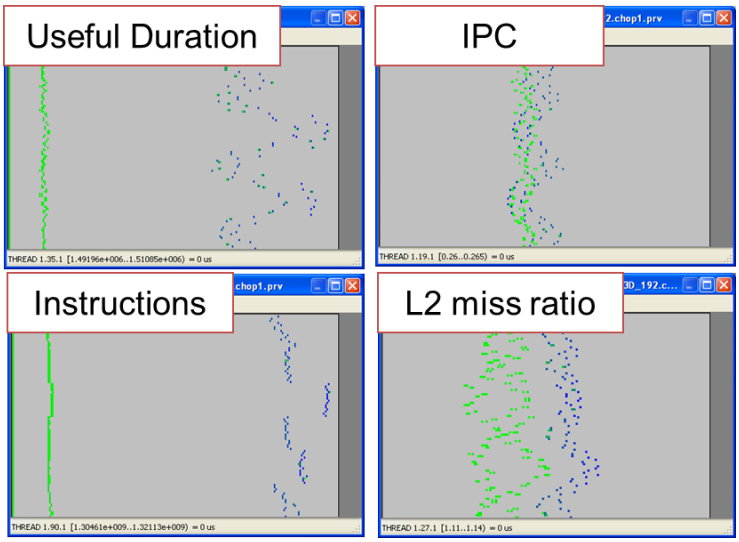

Figure 20: Histograms of different metrics for the SPECFEM3D run

Vertical lines in these histogram plots represent metric instances well balanced across processes (all of them fall in the same bin). Scattered clouds of colored pixels indicate variability in the metric across processes or even across instances for a given process. The wider the cloud, the more variability and/or imbalance it reflects. The duration histogram shows a well-balanced region of short duration (left vertical line in light green) but evidences a significant imbalance in execution time for the long computation bursts. This imbalance happened to be significantly larger than what the developers were aware of. A look at the instruction histogram also shows imbalance that even if not so important it was still larger than what the developer thought. The plots on the right show the existence of variability and imbalance in the L2 miss ratio which directly translates into IPC imbalance. The developers were not aware of this effect, whose combination with the imbalance in instruction happens to result in a very important imbalance in duration.

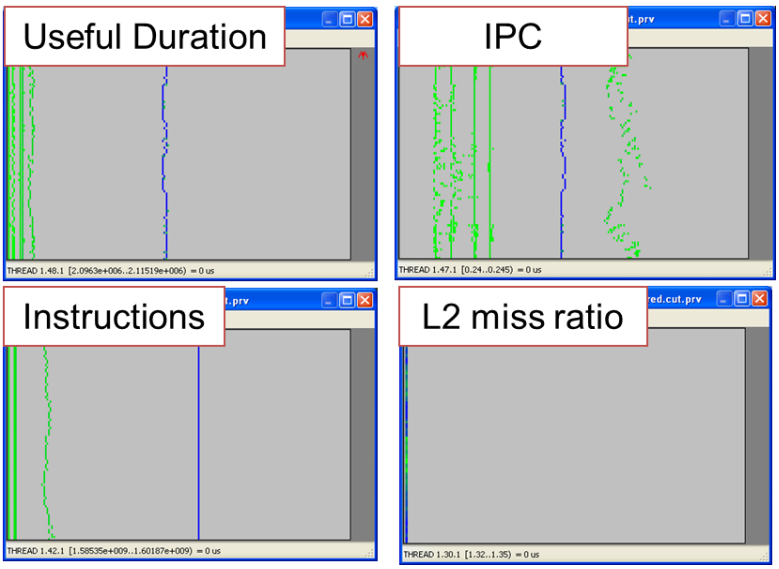

Figure 21: Histograms for optimized version of SPECFEM3D

Figure 21 shows the result after the 6 month effort by the developers on improving the domain decomposition and the numbering algorithm within each domain. The result of this was a much lower L2 miss ratio, while the result of the new domain decomposition is apparent in the vertical blue line in the histogram of instructions. Although not very important in terms of time the light green line in this histogram does show some imbalance. The developers pointed to possible errors in the measurement system as they could guarantee that every process was processing the exact same number of identical elements. Paraver can list where in the source code such imbalance generates and it came out that it originated from trigonometric function calls, the number of which was perfectly balanced as the developer claimed, but whose number of instructions varies as the iterative algorithm used to compute them converges differently depending on the argument. The example does show the huge potential information that histograms provide and how Paraver was useful to drive the development in the most productive direction.

Within the Mont-Blanc project, SPECFEM3D is being ported to a hybrid MPI+OmpSs model. Although the current pure MPI version already uses non-blocking MPI calls and overlaps communication and computation, the use of OmpSs offers an alternative to achieve the same functionality with a simpler source code structure. We have used Paraver to look at the performance of the MPI+OmpSs version, with the aim to identify the potential performance problems in such a version.

Figure 22 shows a zoom of the timeline of tasks executed in a run with 4 MPI processes each of them with two threads. Colored regions correspond to different tasks, while black regions correspond to overhead or waiting time. In this case, the large black regions in process one and four correspond to overheads of creation of tasks while the small holes observed after the grey regions corresponds to waiting time. The green flags are paced at the entry and exit of the task, thus showing regions of different granularities. Some of the tasks have duration in the order of a few hundred microseconds while large task are in the order of a few milliseconds. Zooming in more detail we can even see some tasks taking only a few microseconds.

The flexibility of Paraver allows us to quantify the different effects. The time not executing useful tasks is 13.5% when averaged over the whole run of which the task creation overhead is 3.6% and the idle time due to dependencies 2.5%. There is an important overhead component in the scheduling and switching between tasks which accounts for the remaining 7.4%. Looking at the trace we observe a very large number of traced events within this last region, meaning that the instrumentation overhead can be significant in that phase.

The overall conclusion is that even if minimization of the overheads of the runtime has to be considered, the flexibility introduced by the OmpSs results in programs that are simple to read and perform at the same level as highly optimized and more difficult to maintain pure MPI programs.

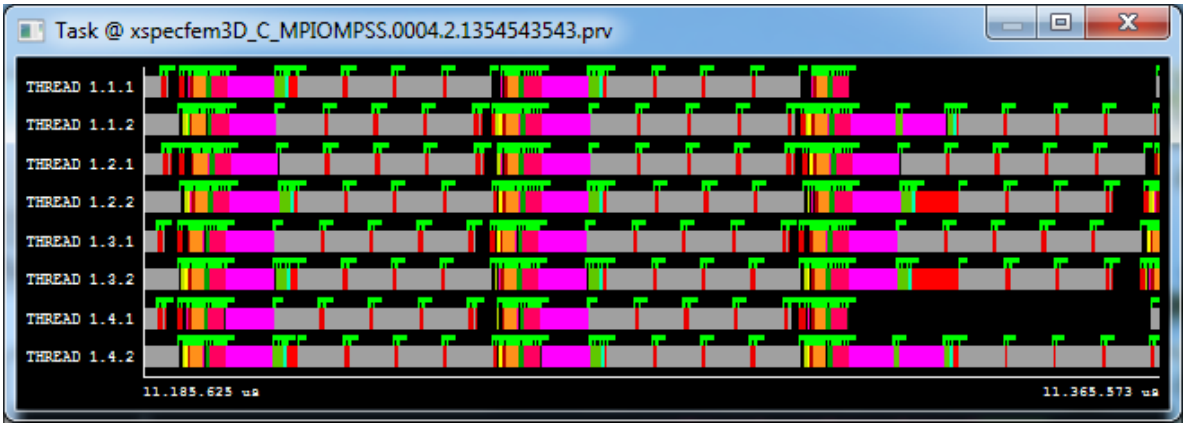

Figure 22: Zoom of the task timeline for a 4 processes each with two threads execution of SPECFEM3D with OmpSs

2.4 EU FP7 CRESTA application suite

The CRESTA project [39] has two integrated missions: one focused on enabling a key set of co-design applications for exascale, the other focused on building and exploring appropriate system software for exascale platforms.

Within the HOPSA project new Vampir tool features have been developed to facilitate performance analysis of complex high parallel applications for exascale. Two simulation codes, GROMACS and Nek5000, have been used to validate Vampir for this great challenge.

2.4.1 Description of the application codes

GROMACS [26] is the major open source and free software package for bio-molecular simulation and developed as an international collaboration steered from KTH, Sweden.

It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions and supports all the usual algorithms expected from a modern molecular dynamics implementation. The traditional strong point of GROMACS has been exceptionally high efficiency, which is also the reason it is used as the engine in Folding@Home.

GROMACS 4.5 uses a multiple-program, multiple-data (MPMD) approach combined with 3D neutral territory decomposition of non-bonded interactions and 2D reciprocal grid decomposition to enable scaling to thousands of cores.

Recent development in GROMACS has enabled scaling to >150,000 cores on the Jaguar XT5 at ORNL for large systems (packages like NAMD also exhibit very good weak scaling), but this is presently only possible to achieve with cut-off methods, and requires system sizes up to 100 million particles. However, even this is turning into a critical bottleneck, since most bio-molecular systems of application interest are far smaller (maybe 500,000 atoms), which puts extremely hard demands on strong scaling. We would like to use 10 million cores to simulate a 500,000-atom system and study dynamics that is 3-4 orders of magnitude slower than what is possible today, which will require completely new disruptive technologies based on adaptive sampling rather than long individual trajectories. The cutting-edge efficiency and open source nature of GROMACS combined with room for parallelisation improvements makes this an ideal CRESTA application.

The first objective for GROMACS in the CRESTA consortium is to incrementally improve scalability by rewriting the code base. In addition to improved FFT algorithms and support for heterogeneous architectures such as CPU-GPU parallelism on each node, algorithms need to be developed that avoid communicating grids over all processors. A second CRESTA objective is to introduce new disruptive approaches for strong scaling in GROMACS. This will be accomplished by using Markov State Models and kinetic clustering for parallel adaptive simulation.

Nek5000 [27] is a highly scalable open-source simulation code for the simulation of incompressible flows in moderately complex geometries. The discretisation is based on the spectral-element method (SEM), which combines the high-order accuracy from spectral methods with the geometric flexibility of finite-element methods. Such a scheme is inherently more accurate than the present industry standard of using low-order finite volume codes. In particular for flow problems involving high-quality DNS of canonical turbulent flows, flow stability and transition studies high order gives clear advantages. There will certainly be a change in industrial codes towards higher order methods, and Nek5000 could be a pioneering method. In particular, different physical models are available, including passive and active scalars, combustion models and large-eddy simulation. Also, tools for flow stability and optimisation are available such as Arnoldi solvers etc.

Nek5000 includes extensive pre- and post-processing software, and filters to various other tools. Nek5000 is maintained at Argonne National Laboratories (USA) by Paul Fischer, and is used at NASA and many universities in Europe, including e.g. KTH Stockholm, and ETH Zürich. The KTH FLOW group has extensive collaborations with the Nek5000 development team.

Since its beginning, Nek5000 was designed to be a code to employ large-scale parallelism. The Nek5000 code has a long history of HPC development, including a 1999 Gordon Bell Award, and successful scaling to 290,000 processors on the Juelich BG/P. The Nek5000 kernel has been fully characterized. Kernel arithmetic is dominated by matrix products and well suited to anticipated exascale architectures. Nek5000 communications are dominated by message latency from one node to a subset of others, although it also uses global reductions.

2.4.2 Description of the performed experiments

The simulation codes GROMACS and Nek5000 have been instrumented and executed on the Lindgren Cluster at PDC, a major Swedish supercomputing centre. Lindgren is a 16 rack Cray XE6 system with a theoretical peak performance of 305 teraflops. The machine is based on AMD Opteron 12-core “Magny-Cours” (2.1 GHz) processors and Cray Gemini interconnect technology.
The measurement of GROMACS was performed with Score-P whereas the measurement of Nek5000 was performed with its predecessor VampirTrace.
Vampir has been used to visualize the execution behavior of GROMACS and Nek5000. Particularly, the "Partial Loading" of Vampir and the "Rewind Option" in Score-P have been validated.

GROMACS

Based on a run of the GROMACS software package with 384 cores (Figure 23) experiments aimed at reduction of events recorded by Score-P by using a new feature called “Rewind” that was developed

within the HOPSA project. This enhancement enables to decide whether or not to record a contiguous section of already measured events during runtime. This dynamic decision may depend on the presence or absence of certain behaviour patterns as well as on similarities or differences with other sections.

To evaluate this enhancement, a frequently executed code section was selected (Figure 20) and manually instrumented so that events of this section will be recorded only if its execution time differs significantly from previous ones. This limits the event recording to exceptional executions of the selected section for later analysis and is expected to reduce the amount of recorded events extensively.

Results of these experiments are presented in the next chapter.

Nek5000

Goal of the experiments with the simulation code Nek5000 was to investigate communication behaviour by means of event-based monitoring. Nek5000, used in the 3D version, has been traced for 1024 MPI processes for 150 iterations.

To visualize and analyse the recorded event data Vampir and VampirServer have been used. VampirServer, performing parallel analysis on 32 processes, was necessary to analyse the large amount of event data.

To enable performance analysis of such large event data volumes on smaller machines, such as desktop machines, Vampir has been extended with the partial loading feature. Partial loading allows visualizing only a specific segment of a trace without the need of loading the complete trace data. Partial loading was developed within the HOPSA project. It enables performance analysis of complex high parallel applications on desktop machines by loading and analysing only hot spots.

Examples of found performance bottlenecks are shown in the next chapter.

2.4.3 Selected results

GROMACS

The analysis of the experiments with the GROMACS software package revealed that using the new Score-P “Rewind” feature reduced the total amount of recorded events by about 60% (Figure 26).

Regions where “Rewind” was applied and hence no events were recorded are presented by a “measurement off” region in Vampir that is color coded in white by default (Figure 25, 26, 27).

Compare views in Figure 26 and 23 also visualize that total application run time and execution behaviour is not affected significantly by the Rewind enhancement.

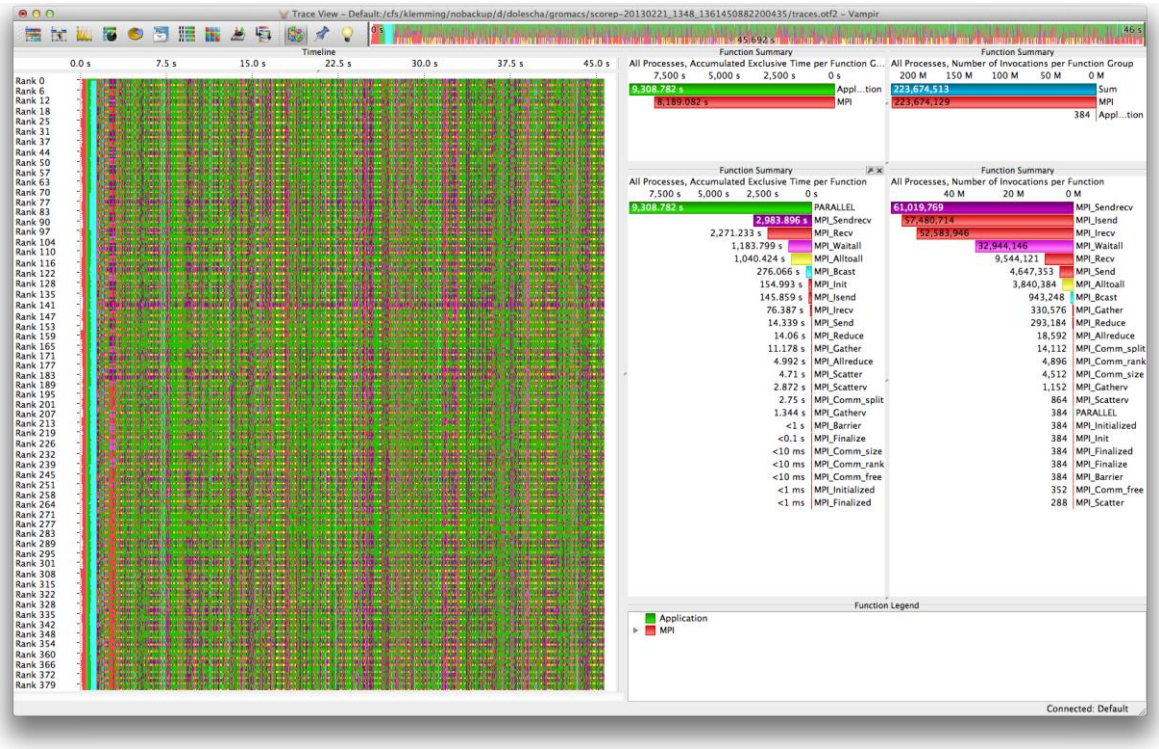

Figure 23: Vampir's Visualization of the Gromacs Trace running on 384 cores

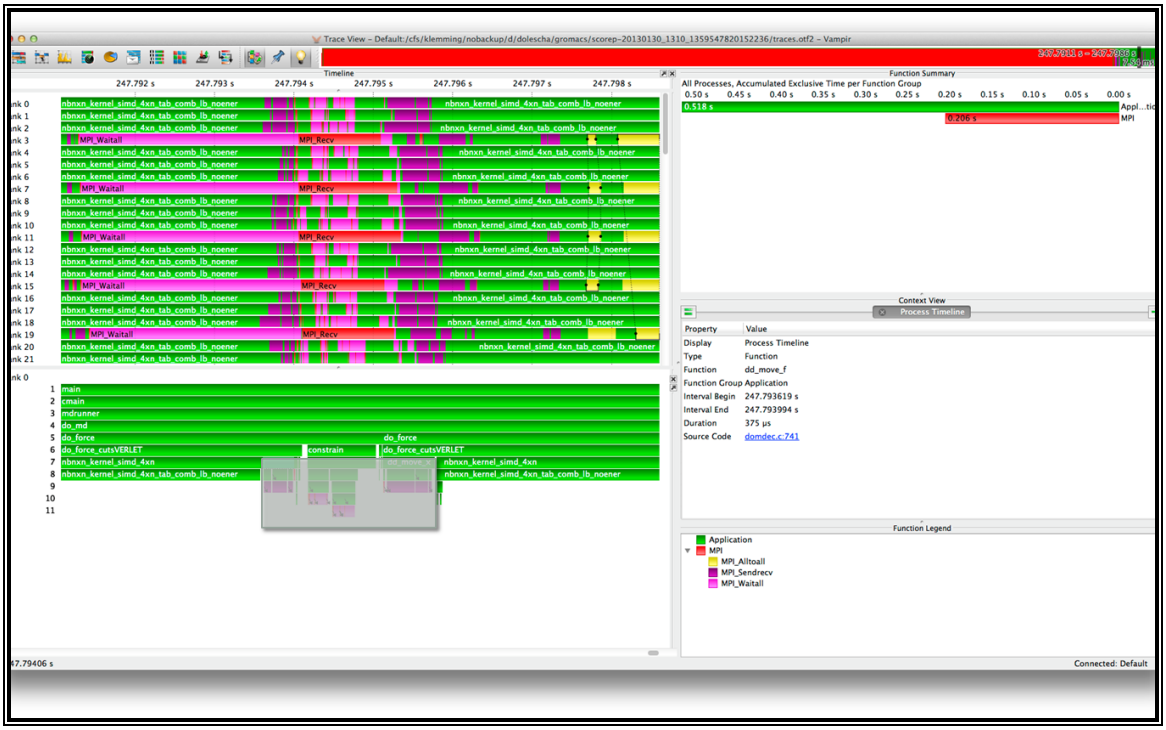

Figure 24: Vampir's Visualization of the Gromacs Trace showing the selected "Rewind" section (grey)

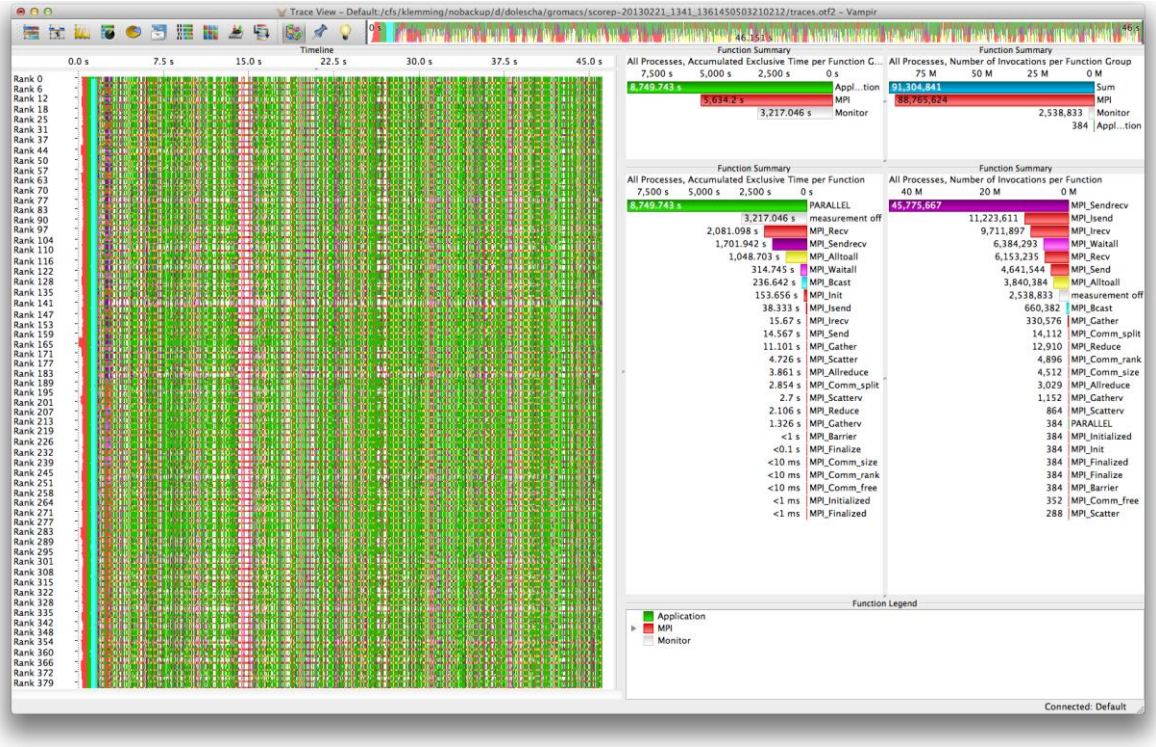

Figure 25: Vampir's Visualization of the Gromacs Trace with the "Rewind" option, "Rewind" sections are color coded in white

Figure 26: Compare View of the Gromacs Trace (original white background, "Rewind" grey background) showing equal runtime

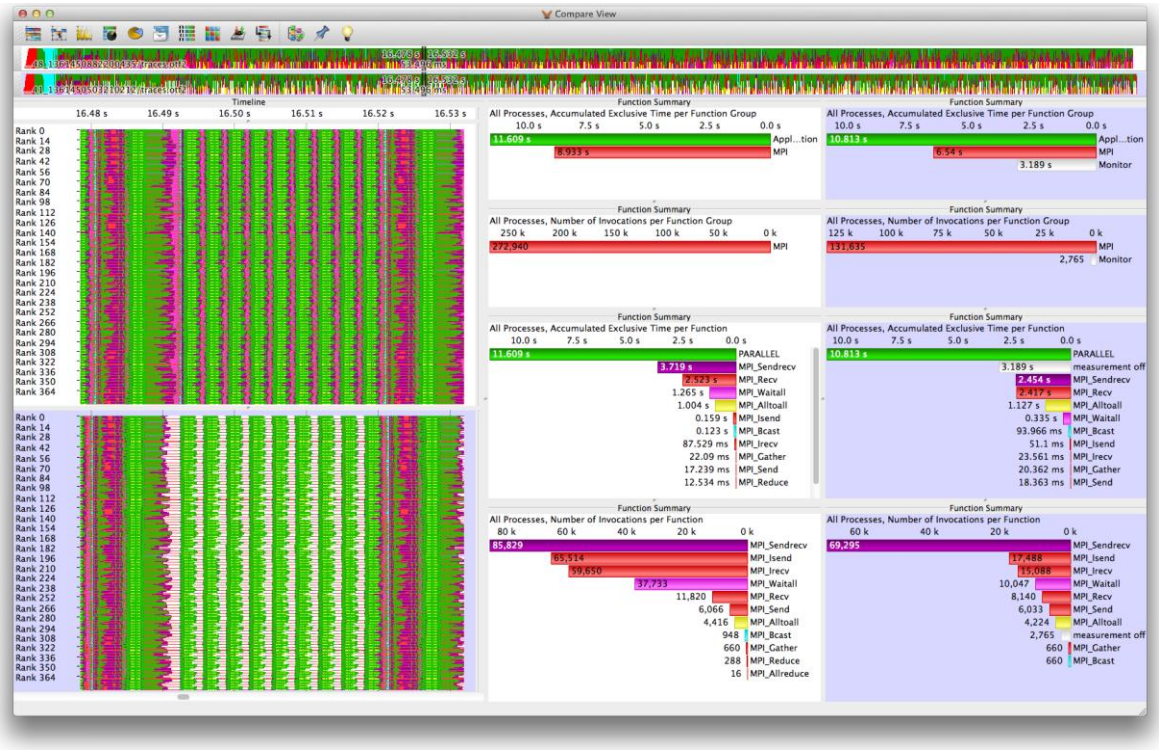

Figure 27: Compare View of the Gromacs Trace (Zoom-in) analogous to Figure 26 showing equal program behaviour

Nek5000

The analysis of the Nek5000 simulation code showed that 13% of the overall runtime is spent in communication routines. The most of this time is spent in 740 million small point-to-point (p2p) messages, resulting in a high time share for *MPI_Waitall* and *MPI_Allreduce*, see Figure 28. This results in a very latency-bounded communication (>25% of all messages are smaller than 1KB, the largest message is 55KB).

The analysis revealed an additional performance problem. The serial I/O operations of process 0 cause a high wait time for all other processes, resulting in a high performance loss, see Figure 29.

Also load imbalances in several computation iterations have been detected, seen Figure 30.

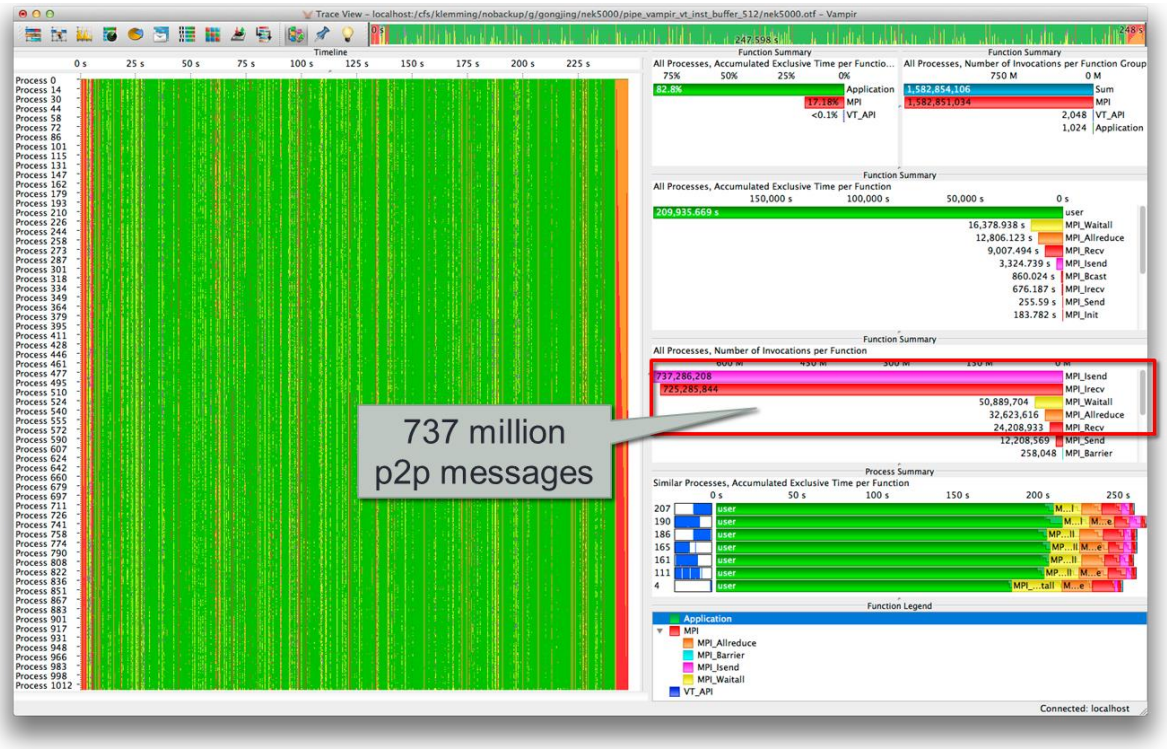

Figure 28: Vampir's Trace Visualization of Nek5000 showing performance problems in communication

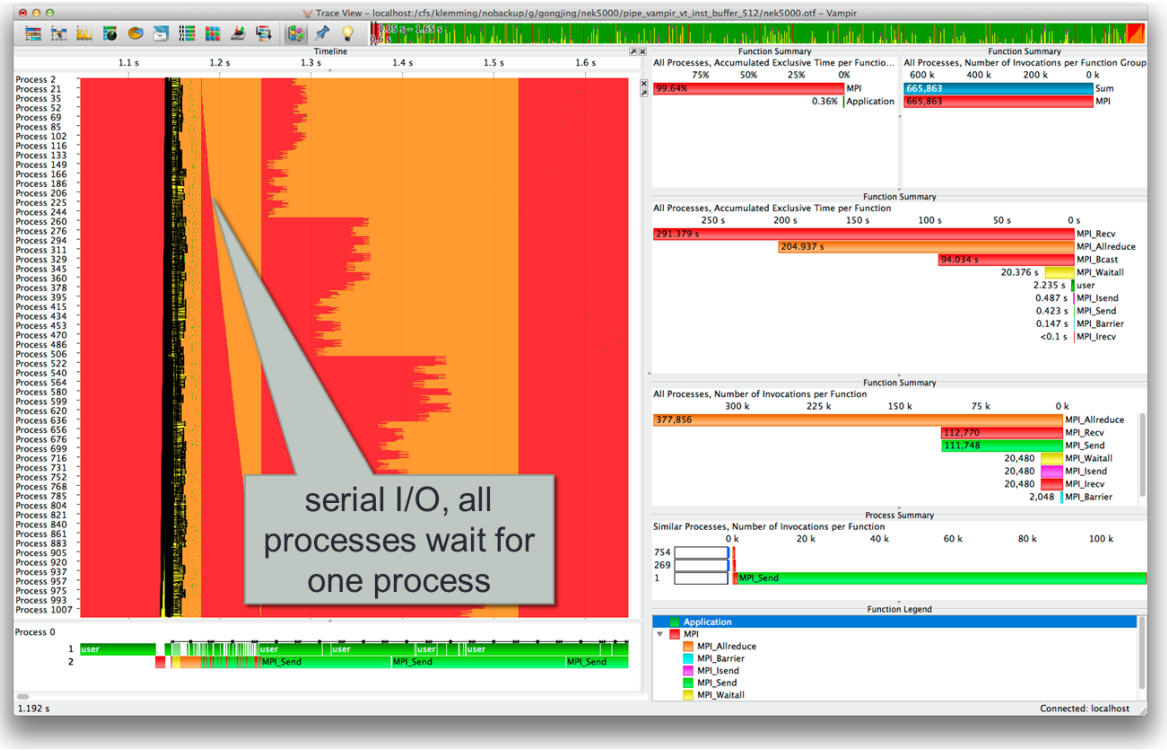

Figure 29: Vampir's Trace Visualization of the initialisation phase of Nek5000 showing a bottleneck due to serial I/O of process 0

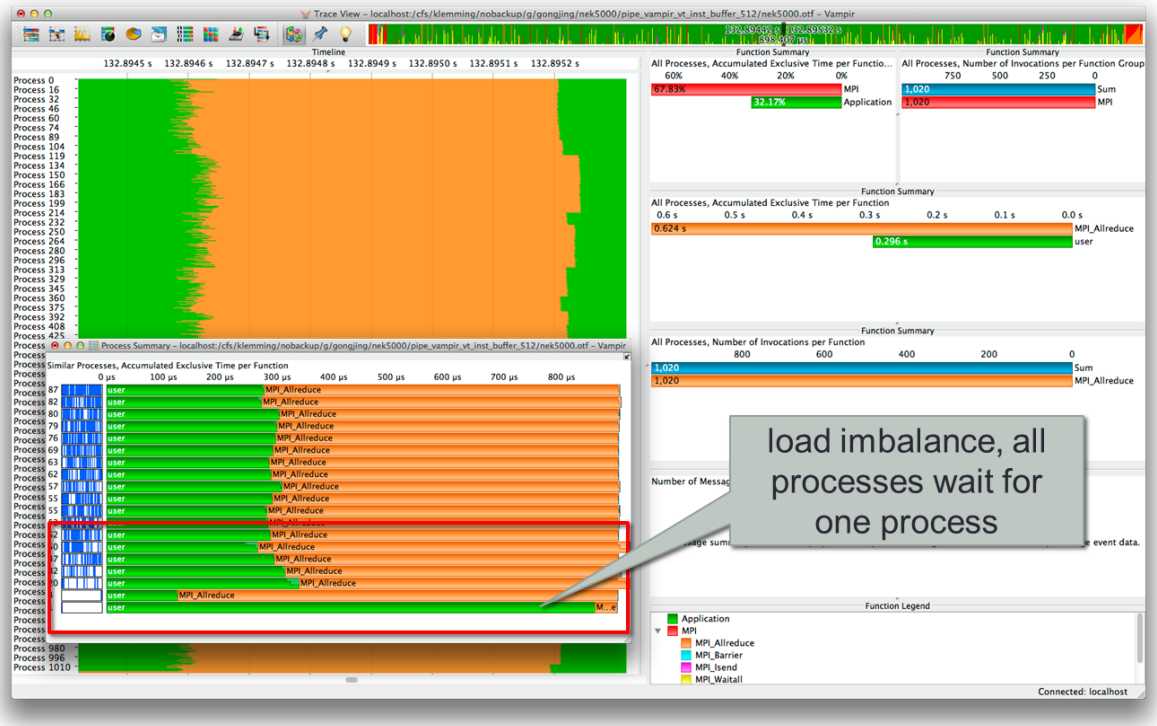

Figure 30: Vampir's Trace Visualization of one computation iteration of Nek5000 showing a bottleneck due to load imbalance

Using Vampir's partial loading feature it is now possible to perform a selective analysis of specific hot spot areas in the trace data. The trace data to be loaded can be restricted in time and space dimensions. A user-friendly trace overview allows selecting an arbitrary time range for loading. A process filter dialog allows loading only processes/threads of interest. Figure 31 shows the partial loading dialog. In the example shown in Figure 31 only the initialisation phase of Nek5000 is loaded. Included are events in the time range between 0 seconds and 1.6 seconds. Figure 29 shows the partially loaded trace in Vampir. Additionally, shown in Figure 31, only processes from 512 to 1024 are involved in the performance analysis. This partial loading process takes about 36 seconds using 4 analysis threads on a quad-core processor. Hence it is possible to perform such an analysis on a commercial desktop machine.

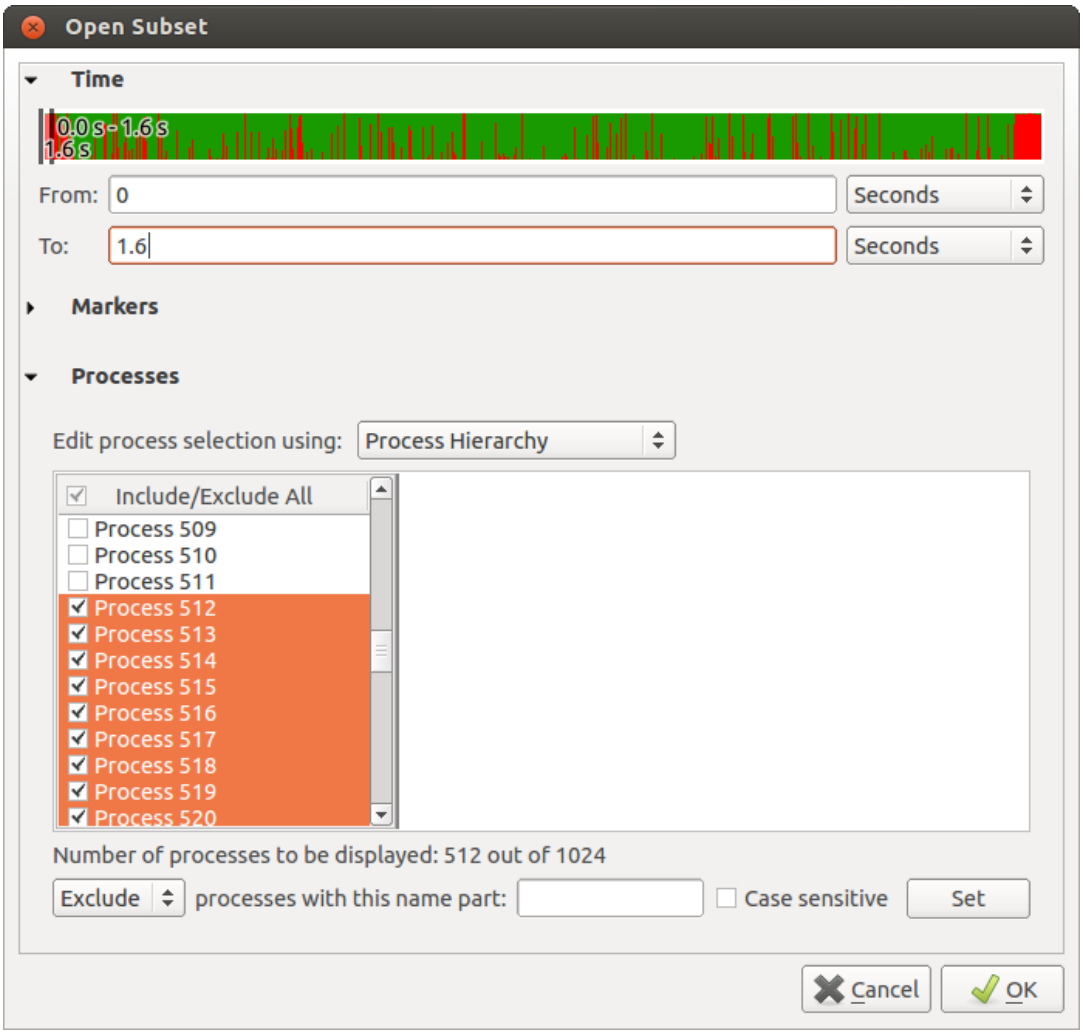

Figure 31: Vampir's Partial Loading Dialog showing the loading of the Nek5000 trace in the time range of 0 seconds to 1.6 seconds for processes 512-1024

2.5 EU FP7 TEXT application suite

HOPSA partners BSC and JSC were also partners (among others) in the EU FP7 project TEXT ("Towards EXascale applicaTions") [40]. TEXT was centered on the vision that the key component to support high productivity and efficient use of a system is the programming model, and that MPI/SMPSs is a hybrid approach that can today be demonstrated and show the way to follow on the path to exascale. The SMPSs model provides the necessary support for asynchrony and heterogeneity as well as enabling incremental parallelization, modularity and portability of applications. The following simulation codes from different fields of science were used to demonstrate the usability of MPI/SMPSs:

- PLASMA (Linear algebra), University of Manchester, UK
- PEPC (Plasma physics), JSC, Germany
- PSC (Plasma physics), JSC, Germany
- SPECFEM3D (Seismic wave propagation), UPPA, France
- BEST (Lattice Boltzmann solver), HLRS, Germany
- MarDyn (Molecular dynamics), HLRS, Stuttgart
- CPMD (Molecular dynanics), IBM Research, Switzerland

The application teams used, in cooperation with BSC and JSC tool experts, the Paraver and Scalasca toolsets to support application analysis and performance tuning. The applications, the experiments performed as well as the results obtained by these experiments are described in detail in the TEXT project deliverable D5.3 (“Optimized Porting of Applications and Kernels”) [24]. As an example, experiments with the PSC code are presented in the flowing sections. A major issue of the code was load imbalance. We show in this sections some of the analyses for which the BSC tools proved very useful in visualizing the behaviour of the different automatic load balancing mechanisms that were implemented within the SMPSs runtime.

2.5.1 Description of the application codes

The Plasma Simulation Code (PSC) is a three-dimensional (3D) electromagnetic Particle-In-Cell (PIC) code that has been re-engineered at JSC to become one of the standard plasma codes in Jülich, readily usable on emerging architectures. The code solves the relativistic Vlasov equation coupled to the Maxwell equations in Cartesian geometry for an arbitrary number of particle species. It can thus be used for state-of-the-art particle simulations in the field of intense laser-plasma interaction, where multi-dimensional computer simulations are now routine due to the highly non-linear nature of the light-matter interaction physics accessible with currently available laser technology.

PSC was a MPI parallel code that is highly synchronous between the different MPI ranks. Each time step requires several exchanges of fields and particles between nodes to carry on with the computation. Due to the fact that PSC is a mesh-based code, load-balancing is non-trivial and can become an issue if the plasma (particles) to be computed is spatially very localised. The rectangular domain decomposition can try to adjust column width and row height to account for different workloads. However, situations are possible where some processes end up with a very small computational load (only from computing the fields) while others have a high computational load (a similar field volume but also many particle updates).

2.5.2 Description of the performed experiments

PSC was ported to MPI+SMPSs within the TEXT project. Two initial objectives were to overlap communication and computation and to enable the automatic load balancing by means of the DLB library [14]. Work on the analysis of such an application and the interaction of load balancing mechanisms and reductions have been performed after the end of the TEXT project. We report here some of the results of the analyses done with Paraver and Scalasca to understand such interaction and provide guidance to improve the future support of reductions in hybrid models.

2.5.3 Selected results

While developing DLB we observed some instabilities and strange timings that we associated to thread migrations by the linux scheduler. In order to verify that hypothesis we used the Extrae API to inject user events into the trace. At the beginning of each task we would find out the core where the task was running and emit it as an event into the trace. Paraver could then visualize for each task the core it was running on. The behaviour with a simple micro-benchmark is shown in Figure 32 where each core appears as a different colour. The figure on the left corresponds to the original DLB behaviour and it is evident that several tasks are sharing the same core at some points in time. This was an undesired result. We had to modify the runtime to better control the pinning of threads to cores and handoff of the cores between appropriate threads in order to avoid the over-commitment of cores. The view on the right side shows the behaviour of the optimized runtime which is more regular and improves the performance by more than 15%.

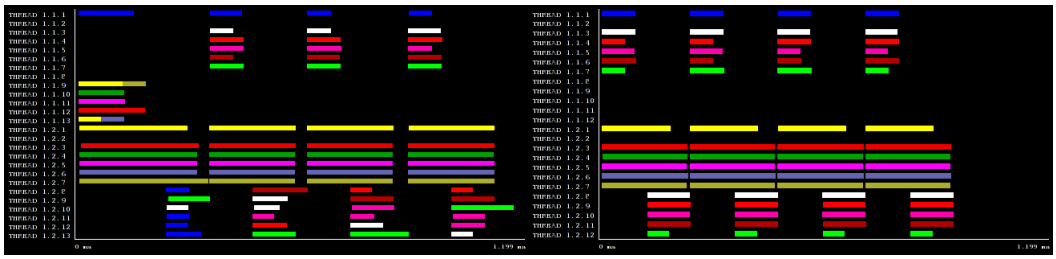

Figure 32: Timeline showing the core being used by each task. The view on the left shows pinning bugs in the implementation where several threads share the same core which were fixed for the view on the right.

Figure 33 shows the behaviour of two iterations of the PSC code under DLB. We see in purple (push_particle_2d) the tasks computing the contribution of the particles to a privatized copy of the main array on which indirect (very scattered) accumulations are done. The green (reduce) tasks correspond to the accumulation of those contributions to the global array. Because of the very large imbalance at the MPI level, some processes do not perform any local contribution and can immediately lend their cores in the push_particle_2d and reduction phase, while other processes start that phase with 4 threads but finalize with 8. The figure corresponds to an optimized version where the number of reduction tasks matches the maximum number of active threads in the push_particle_2d computation. Although simple once understood, this optimization would have not been identified without the visualization functionalities and allowed an important reduction in the total number of instructions executed.

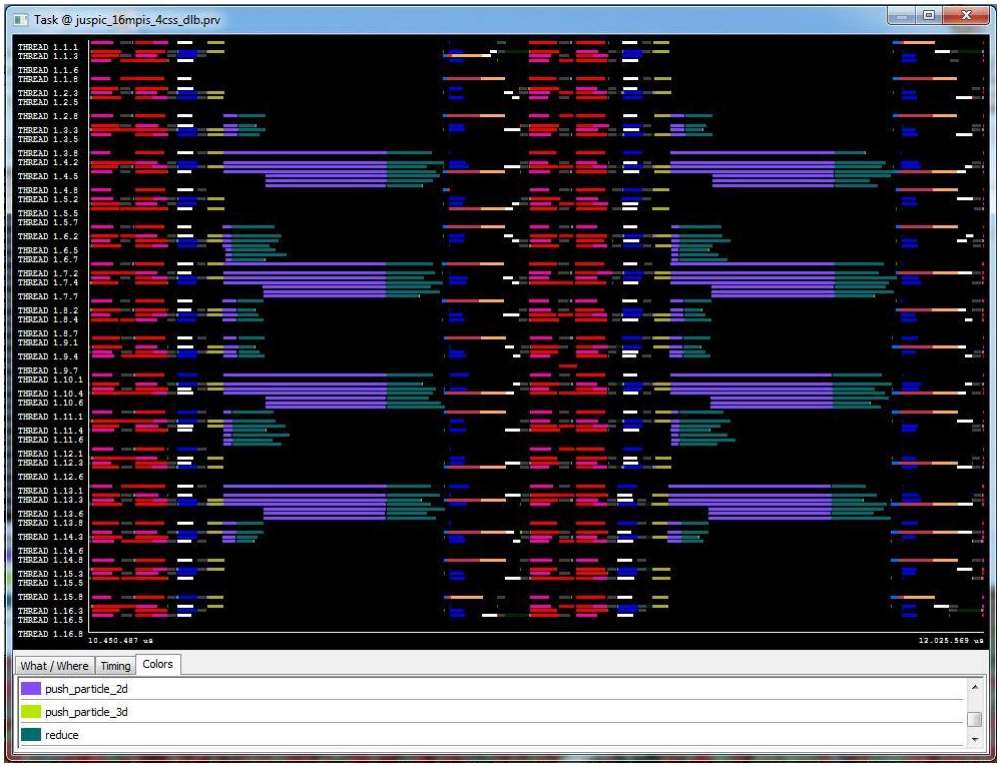

Figure 33: Impact of DLB on the PSC code

The visualization of the behaviour of the hybrid MPI+SMPSs application helped us better understand the interaction between automatic load balancing and the implementation of reduction operations, which will result in better mechanisms being integrated in OmpSs.

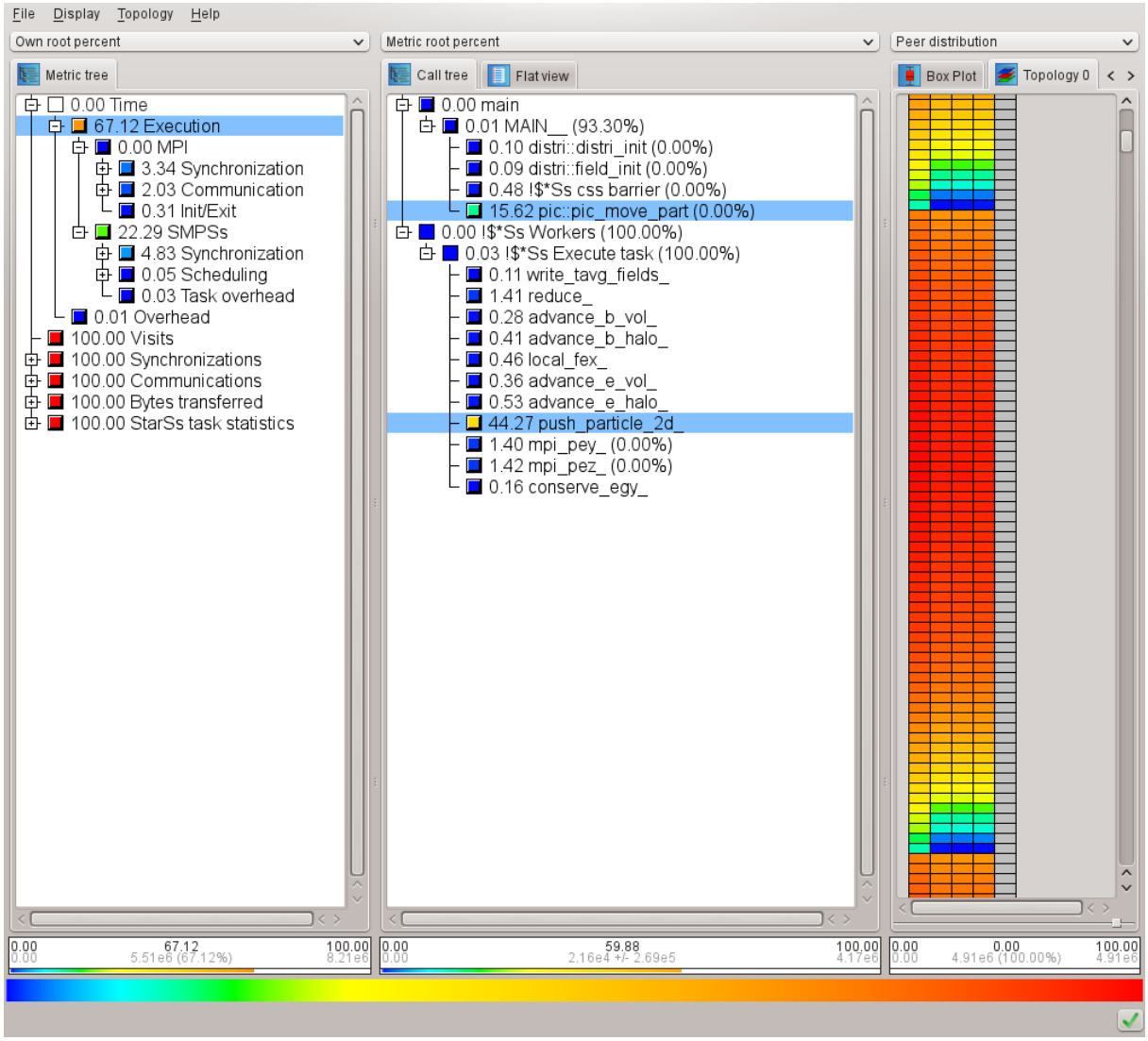

Figure 34: Scalasca analysis of the PSC code running with 2048 MPI processes each with 4 SMPSs threads. Distribution of the execution time of the two most time-consuming SMPSs tasks over the MPI ranks (vertical axis) and threads (horizontal axis) can be seen in the right panel.

As shown above (and in other examples in this report), the timeline views of Paraver and Vampir nicely visualize the dynamic behaviour of an application over time and space (processes/threads). For large-scale and long-running experiments, the trace files necessary for the visualization quickly can become very large, complicating the handling and usability quite a bit. However, scalable profile measurements still can provide useful insights into the application behaviour. Figure 34 shows the profile analysis of a PSC in MPI/SMPSs mode; executed with 2048 MPI ranks each running using four threads for SMPSs task execution. One can easily see that about 25% of the execution time is spent inside the SMPSs runtime, leaving about 75% for user code and MPI. The 67.12% spent on executing user code is mainly spent in two SMPSs tasks (push_particle_2D and pic::pic_move_part). As explained already above in the Paraver analysis, the load balancing is not perfect, as can be seen in the distribution of the execution time of these two tasks over the MPI ranks (vertical) and threads (horizontal). The workload is distributed in blocks to a set of MPI ranks. While the processes in the first three quarters of each block are very busy (red), the last quarter has less to do. Note that not all processes are shown; die screen dump in Figure 34 shows about 80 ranks or 1.25 of the 16 blocks.

2.6 Other applications

The CG-POP code from the earth sciences area used within the G8 ECS [28] project also has been analysed with Paraver. In addition, the CESM sea ice model [29] was analysed with Scalasca. This is described in Section 3.1.2.

2.6.1 Description of the application codes

CG-POP is a miniapp of the POP application. The Parallel Ocean Program (POP), developed at Los Alamos National Laboratory, is an important multi-agency code used for global ocean modeling and is a component within the Community Earth System Model (CESM). The motivation for creating a miniapp for the POP developer team is that it will enable them to ensure the performance portability of the most critical portion of the application while also testing new programming models. The CGPOP miniapp is the conjugate gradient solver from LANL POP 2.0, which is the performance bottleneck for the full POP application. The CGPOP miniapp is written in Fortran90 with MPI and is about 3000 source lines of code (SLOC), whereas the POP application is 71,000 SLOC.

2.6.2 Description of the performed experiments

The goal of the CG-POP study reported on this deliverable was to analyse the performance of the sequential code regions to detect potential optimizations. The folding mechanism allows to obtain a very detailed evolution of the performance metrics with a very low overhead [31][32]. It has been demonstrated as a very useful mechanism for the kind of analysis we target to carry out with CG-POP.

The executions were done in MareNostrum BSC supercomputer where we can use the CPI-Stack model from IBM [33][34] that breaks down the CPI (Cycles per Instruction) on the cycles stalled on the different processor units. The target for the analysis was the MPI21D version of the code, and as the goal was the sequential code analysis the execution was done with 24 MPI tasks, i.e., the minimum number of processes. With such configuration, the block size required is 180x120. For this analysis we did not cover different block sizes because previous analysis demonstrated that smaller block sizes are used to balance the computation when the number of tasks is increased.

2.6.3 Selected results

The CPI-Stack analysis for the main computation phase of CG-POP is collected in Figure 35. The black line corresponds to the achieved instantaneous MIPS and the colored region to the CPI breakdown.

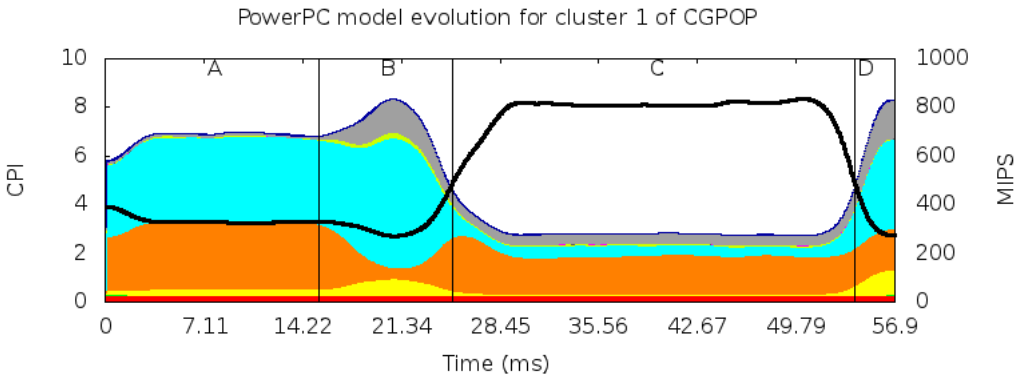

Figure 35: MIPS and CPI breakdown for the main computation phase of CG-POP

We have marked the four internal phases detected by this analysis. Phase C achieves a good performance, while phase B is the one with bigger CPI. Summarizing the color scheme used on the CPI-Stack plot, orange corresponds to LSU (Load Store Unit) other reject, light blue to LSU Data cache misses and grey to the floating point unit basic latency. Correlating the four phases detected with the source code (left code of Figure 35) we identified potential improvements which were very easy to implement:

- Phase B does a floating point reduction and the problem is that the delay to compute the floating point operation cannot be overlapped within the iteration execution, so the latency of the FPU stalls the execution (as reported on the CPI-Stack).. The solution is to merge the loop of phase B with the loop of phase A so the processor can do other computations while the FPU is working.
- As phase D may present a similar problem than phase B, we can apply a similar solution as phase C and D can be easily joined on a library call.

These very simple changes that are reflected on Figure 36 obtained 11% of improvement. Figure 37 compares the CPI breakdown of the original code with the optimised version. Considering that CG-POP is a well optimised mini-application, this percentage is a clear success. When we presented the results to one of the developers of CG-POP he was impressed with the improvement as the tool was able to suggest changes he had never thought about.

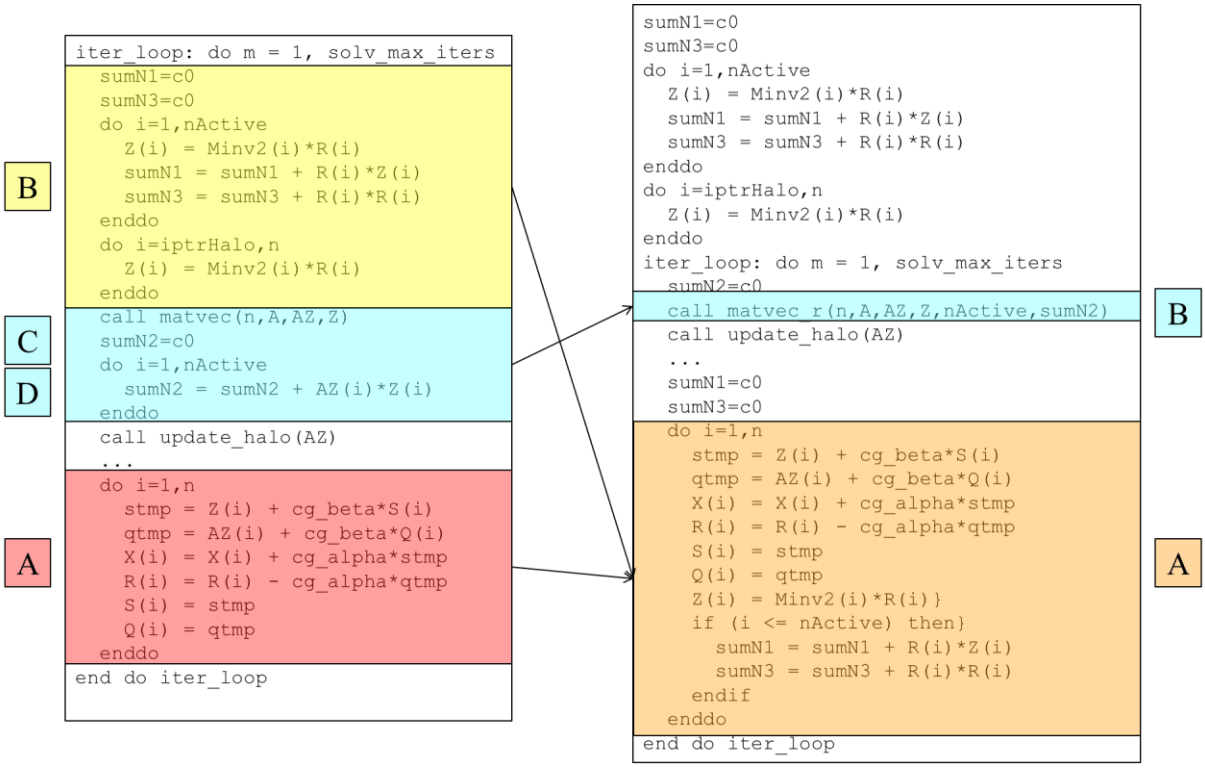

Figure 36: CGPOP Code restructuring

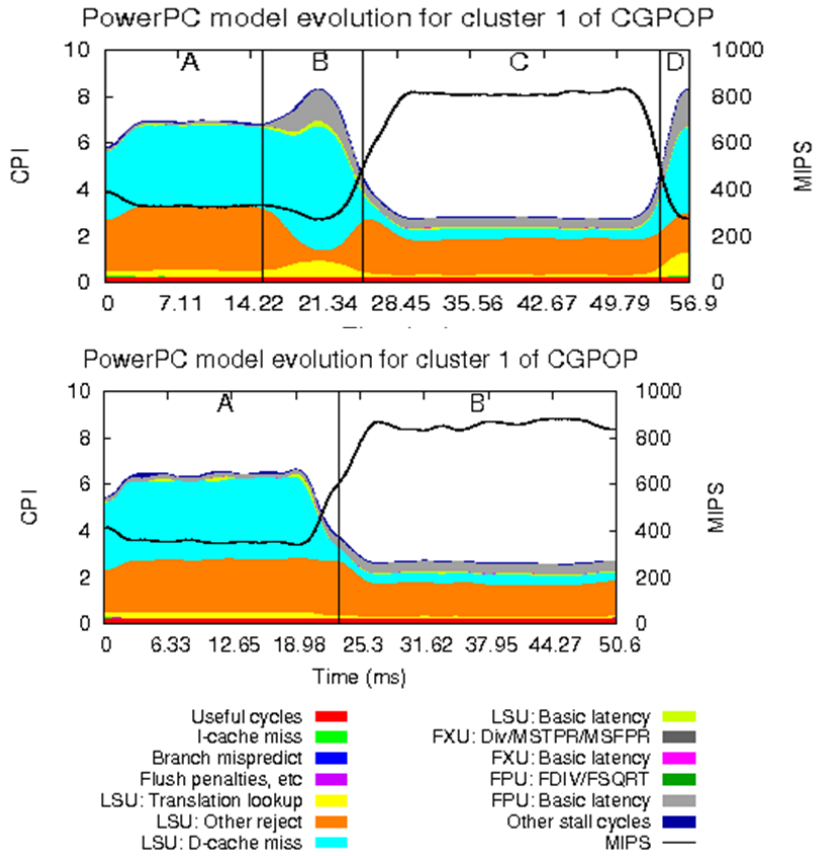

Figure 37: CPI breakdown comparison

3. Selected use cases

In this chapter, we describe experiments and tests which demonstrate that the goals of enhancing the functionality and scalability of the HOPSA tools in the course of the project have been reached and were successful.

3.1 New tool features developed within HOPSA

3.1.1 BSC Tools

From the list of enhancements implemented in BSC tools within the framework of HOPSA we have selected the functionalities that can be better illustrated in a written report through an example of its usage within the analysis process.

Combined system and application analysis with Paraver

System-related performance metrics provided for example by LAPTA [11] complement application event data. One of the difficulties and at the same time advantages of such information is that LAPTA reports data at system level instead of application level. The system sensors information is collected in different ways, but all of them have a granularity not finer than 1 second to control the volume of data generated. We consider this granularity is too coarse for HPC applications where most communications are in the order of microseconds. For that reason we requested to the Russian partners to implement a finer granularity in what they call historic buffer (a circular buffer of 300 entries storing the last values for the sensor). Although our proposal was initially accepted, they have not been able to implement it due to time/effort constrains. With the current coarse granularity, the correlation with the application becomes more difficult.

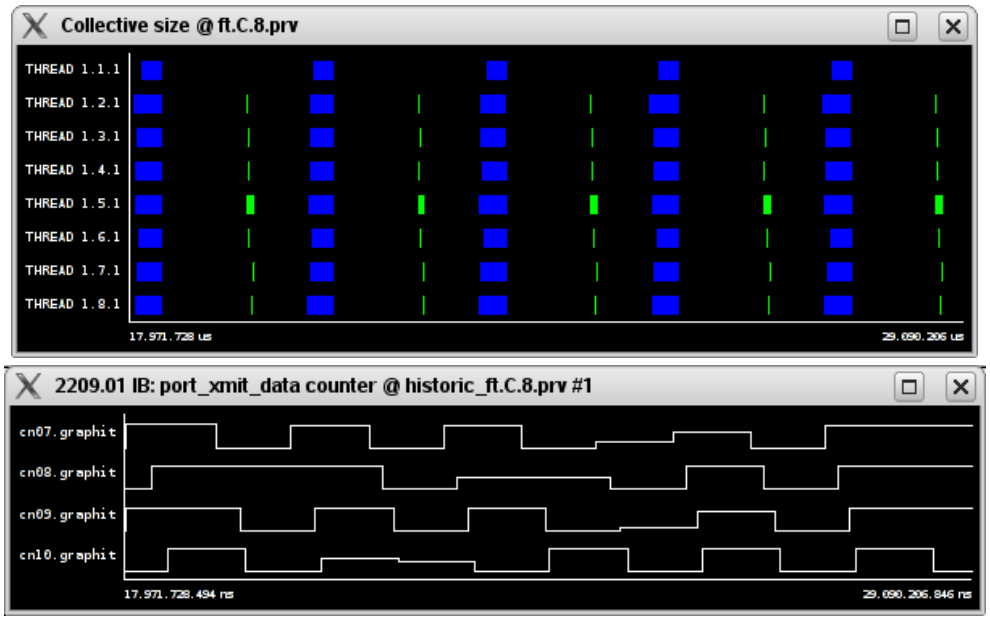

Figure 38: Application view of the collective size and System view of the Infiniband transmitted data counter (both views are synchronized displaying the same time interval)

To validate the benefits of including the system sensors with coarse granularity on the application analysis, we selected the Infiniband network traffic sensors and used them to analyse the NAS FT benchmark v3.3 Class C with 8 processes that alternates *MPI_Alltoall* calls of 32MB with *MPI_Reduce* calls of 16bytes. We instrumented a first run placing two tasks per node and tried to correlate the application communications with the sensors activity. The upper part of Figure 38 shows the size sent by the communication primitive while the bottom part reflects the Infiniband sensor value.

As we expected, depending on the application traffic and the length of the communication phases, it may be complicated to detect whether there was network congestion unless it is maintained for a high percentage of the one second measurements. What we tried to validate is how useful it can be to look at the network sensors when the congestion originated outside the application (by other jobs / processes). To analyse this scenario, we executed the same configuration of NAS FT but this time concurrently with a network greedy program that randomly generates point-to-point traffic that severely stresses the network. We mapped one task per node, on the same nodes as the NAS FT application. The randomness of the network greedy program is with respect of the message size (from 238MB to 1GB), the length of the communication phase and the distance between communication phases (both of them between 4 to 12 seconds).

Figure 39 shows the interfered execution. The first timeline corresponds to the duration of the MPI calls for the NAS FT. As the scale has been set to the duration of the *MPI_Alltoall* primitive we can start to identify the variability between calls. The middle timeline corresponds to the Infiniband network sensor used on the previous experiment, where 4 phases with high traffic are identified in blue and green colors. The bottom timeline corresponds to the network greedy program and displays its communication size. We can clearly correlate that the high network traffic regions detected by the LAPTA sensor correspond to the communication phases of the greedy network program.

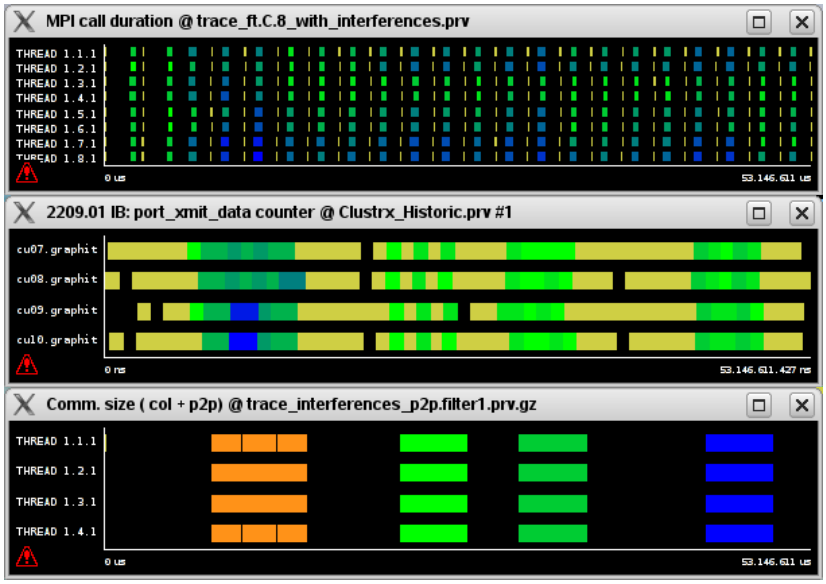

Figure 39: Perturbed execution: Application MPI calls duration, LAPTA Infiniband sensor and Network greedy program communication size.

To measure the impact of this network interference in the NAS FT, we compared the histogram of durations for the *MPI_Alltoall* call. Both histograms are at the same scale, indicating that the perturbed execution had some calls to *MPI_Alltoall* that last longer (Figure 40). If we select the area of colored cells on the rightmost part of the perturbed execution that did not appear on the first run, we can validate with the sensor view that most of them occur during the high network traffic regions detected by the sensor (Figure 41). We can also see that the larger calls (darker blue) happened on the phases with more traffic. This study demonstrated that even with coarse system information at one second granularity, the sensors can provide useful information for the application analysis as long as the effects are maintained long enough.

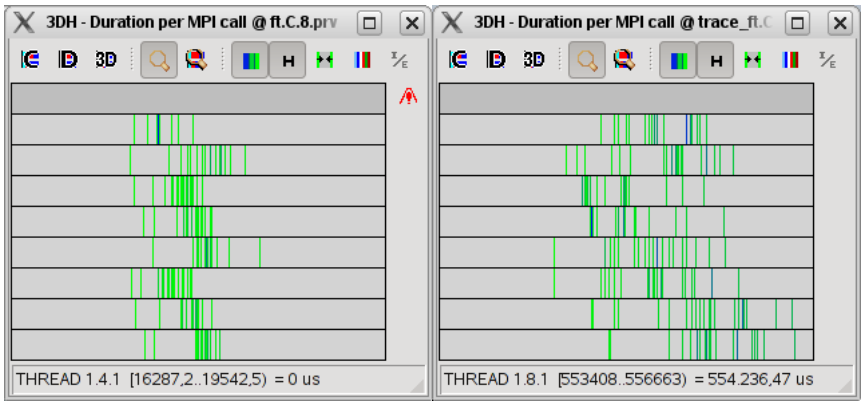

Figure 40: Histogram of the MPI_Alltoall duration. On the left the non-perturbed run, on the right the perturbed run.

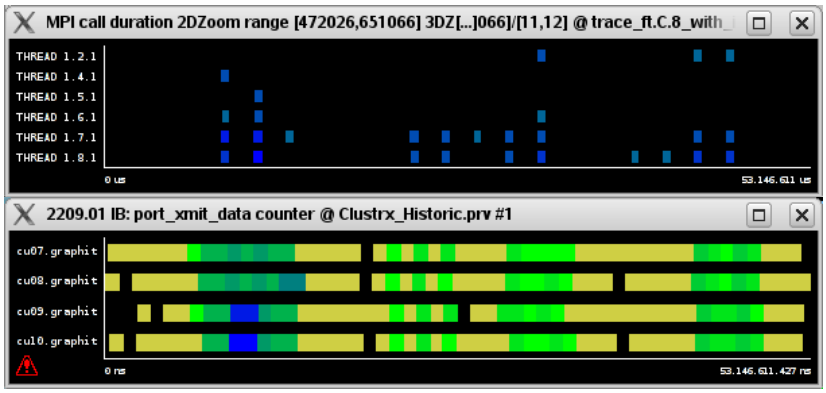

Figure 41: Correlation of the large MPI_Alltoall calls with the high traffic regions identified by the Infiniband LAPTA sensor

Programming models support

Within the framework of the HOPSA project, Extrae has been extended to support accelerators (CUDA and Intel MIC) and new versions or extensions of OpenMP (new Intel compiler runtime and OpenMP tasks) as well as improve the OmpSs support. As an example, in this section we present a brief analysis of the HydroC application that allowed detecting a bug on its porting to CUDA.

Figure 42 compares the execution of one iteration for the pure MPI version with the MPI+CUDA version. The top image corresponds to the main user functions of the MPI version; the second image is the user functions of the MPI+CUDA version for an equivalent duration. Finally the bottom image corresponds to a zoom of the beginning of the MPI+CUDA iteration.

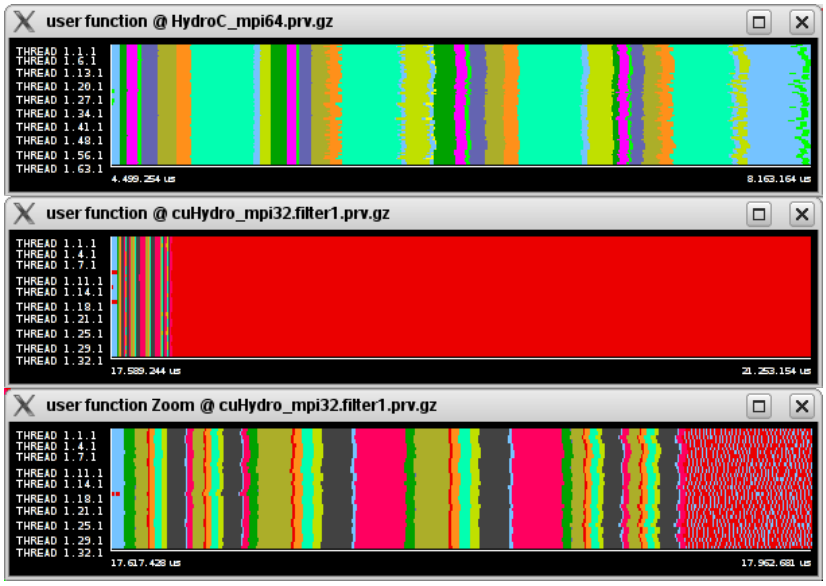

Figure 42: Comparing MPI and MPI+CUDA versions iteration at the level of user functions

As we can see the MPI+CUDA version not only does not reduce the iteration time but it is increased (the capture does not include the full iteration). We can also see that the structure at the beginning of the MPI+CUDA version (zoom) is quite similar to the pure MPI structure with a significant performance improvement (comparing the first two captures). This simple analysis indicates that the problem is on the last call of the red function that dramatically extends its duration (does not happen on previous calls within the same iteration).

Using the CUDA runtime instrumentation we can correlate the user functions with the execution of the CUDA kernels as shown in Figure 43.

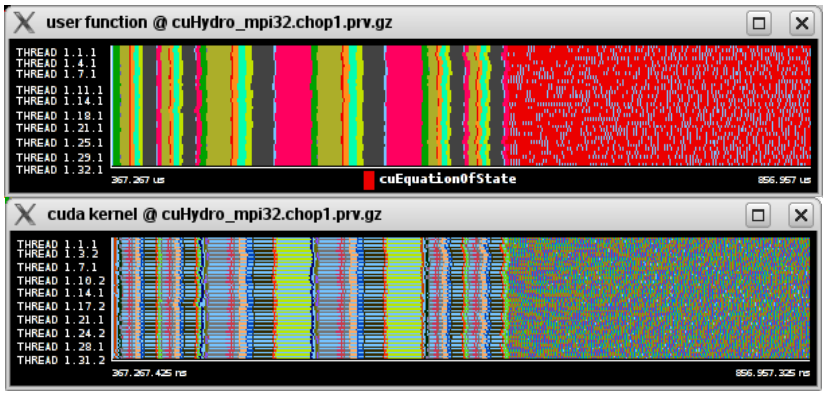

Figure 43: Correlating the user functions with the CUDA kernels

We can also correlate the CUDA kernels with the CUDA runtime instrumented events. In Figure 44 these views are zoomed on few processes and a very short time interval within the problematic region. The top capture displays the kernel from both the CPU and the GPU point of view. At the CPU level (1.*.1) the colored regions refer to the CUDA launch runtime call, while at the level of the GPU (1.*.2) the colored regions state to the actual kernel execution. On the bottom capture we display the runtime calls where white corresponds to CUDA configure call, blue to CUDA launch, pink to CUDA thread synchronize and light blue outside the CUDA runtime calls. On this view there is no information on the GPU lines because these events are only applicable to the CPU.

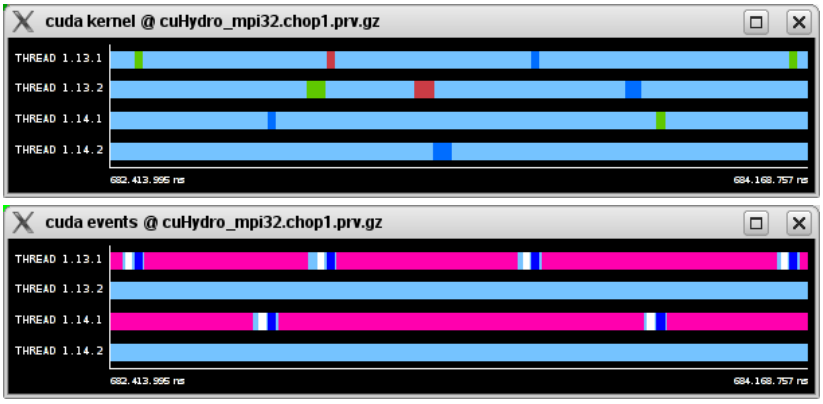

Figure 44: Correlating the CUDA kernels with the CUDA runtime

As this very fine granularity the interference of the instrumentation may have been high. Even if that is the case, the collected information shows us that these kernels are really small and the overhead to execute them in the CUDA device is much larger than the benefit of running them on the accelerator.

When these results were presented to one of the developers that ported the code to GPUs he quickly understood what was happening: the function EquationOfState traverses a matrix structure in its two different dimensions, while in one of them each kernel computes one line, on the other dimension there was a kernel call per cell. He did not realise the impact of that difference until he saw it reflected by the Paraver analysis.

Integration of Cube in the BSC Folding analysis

The folding mechanism takes advantage of the application repetitiveness to provide detailed evolution of performance metrics by combining information provided by coarse grain sampling and instrumentation with the consequent saving on the sampling overhead. The folding mechanism uses instrumentation to identify the regions of an application and the sampling mechanism to detail the evolution within those regions in terms of performance counters and source code references. As long as the application runs long enough, applying the folding to coarse grain sampling can provide similar results for the repetitive regions as the detailed sampling. The folding itself simply gathers the samples of a region spread through the tracefile and maps them preserving their relative time with respect to their region into a synthetic region[31][32]. As a result of this phase, different plots are generated characterizing the performance of the regions with different hardware counters and correlating the phases and regions with the source code trough call stack information.

In this section we introduce the methodology to use Cube to explore the Folding analysis results when applied to OpenMX 3.6p1. OpenMX [35] is a solver for electronic structure based on quantum mechanics principles. The input used is based on code performance testing for PRACE's Curie machine: they represent a material surface (in 4-layer slab geometry) with surface area simply growing in size. By looking at the results shown in the CUBE visualizer, our suggestion is to start looking at the weight of the different phases detected. This view corresponds to Figure 45 where the user can easily identify the most representative regions are Cluster 1 and Cluster 2, so they will be the target of our analysis.

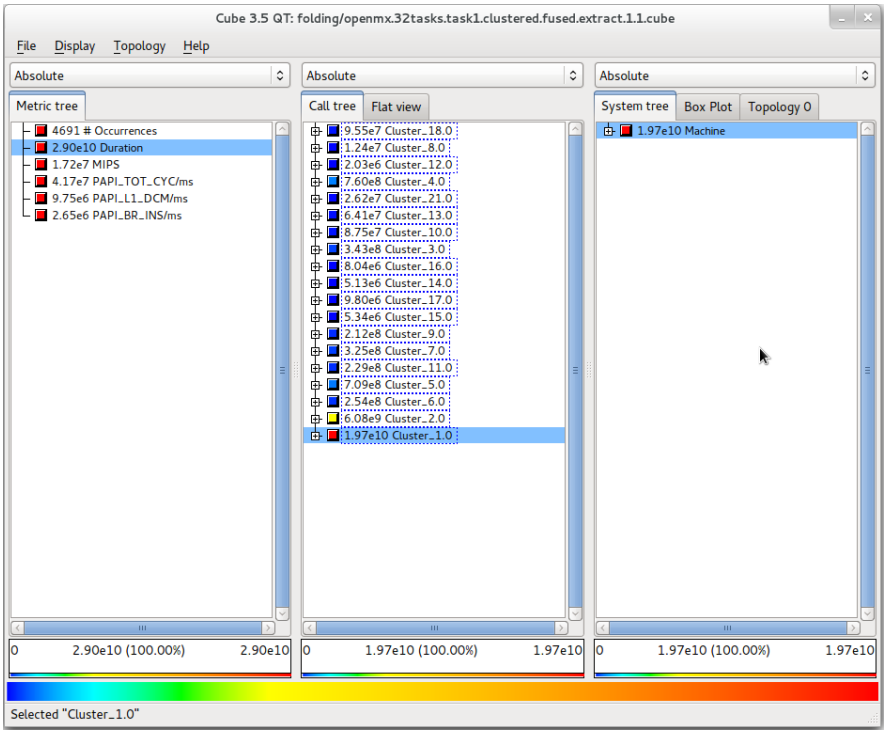

Figure 45: Folding analysis with Cube – detecting main phases with respect to their duration

Typically the second check will be to look at the MIPS metric to evaluate the performance of these regions. This capture corresponds to Figure 46.

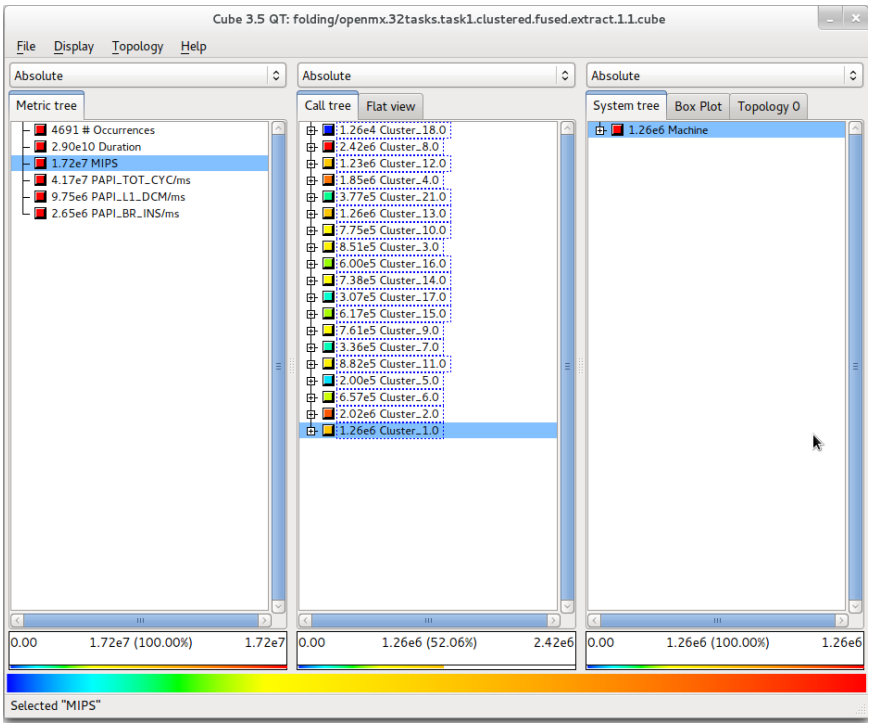

Figure 46: Folding analysis with Cube – Performance evaluation using MIPS

The performance of Cluster 1 is 40% lower than the performance achieved by Cluster 2, so we can look at the internal structure of Cluster 1 by selecting within this node the contextual menu option See detailed MIPS (Figure 47).

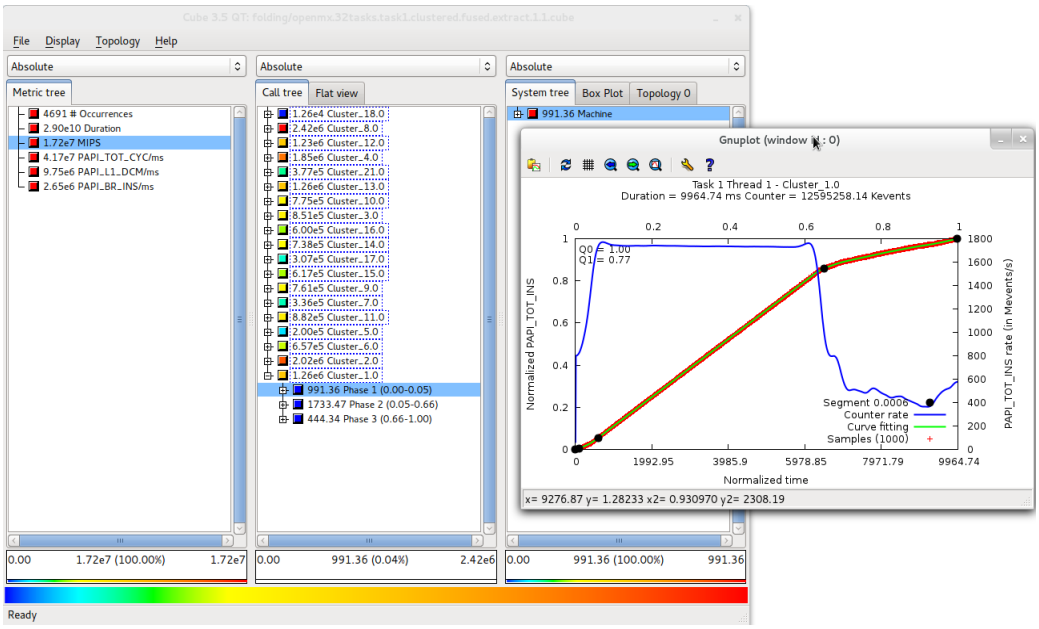

Figure 47: Folding analysis with Cube – Cluster 1 internal phases performance

The launched gnuplot indicates that Cluster 1 region is divided in two phases and the performance problems are on the second part. Selecting the Level 1 Data Cache misses per millisecond metric (PAPI_L1_DCM/ms) (Figure 48) we can see that Cluster 1 region has a large increase of L1 misses on this second part so the next step is to look at the sources and check how to improve the L1 access on this region of the code.

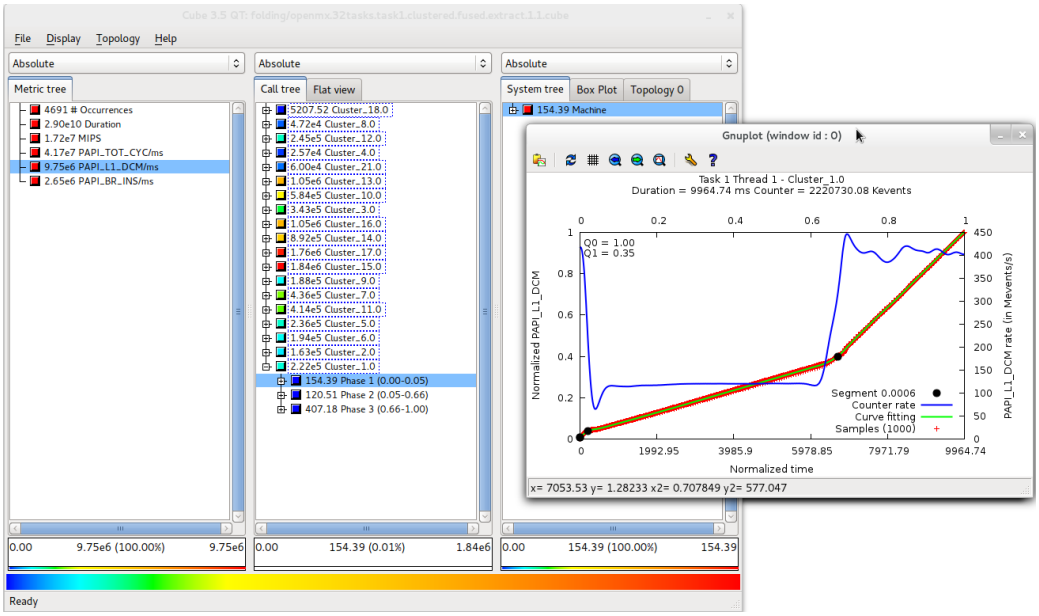

Figure 48: Folding analysis with Cube – Cluster 1 L1 Data Cache Misses

Finally, there is a summary view of all the metrics normalised that can be launched from the Occurrences metric. Figure 49 is an example of this view when we select it for Cluster 2.

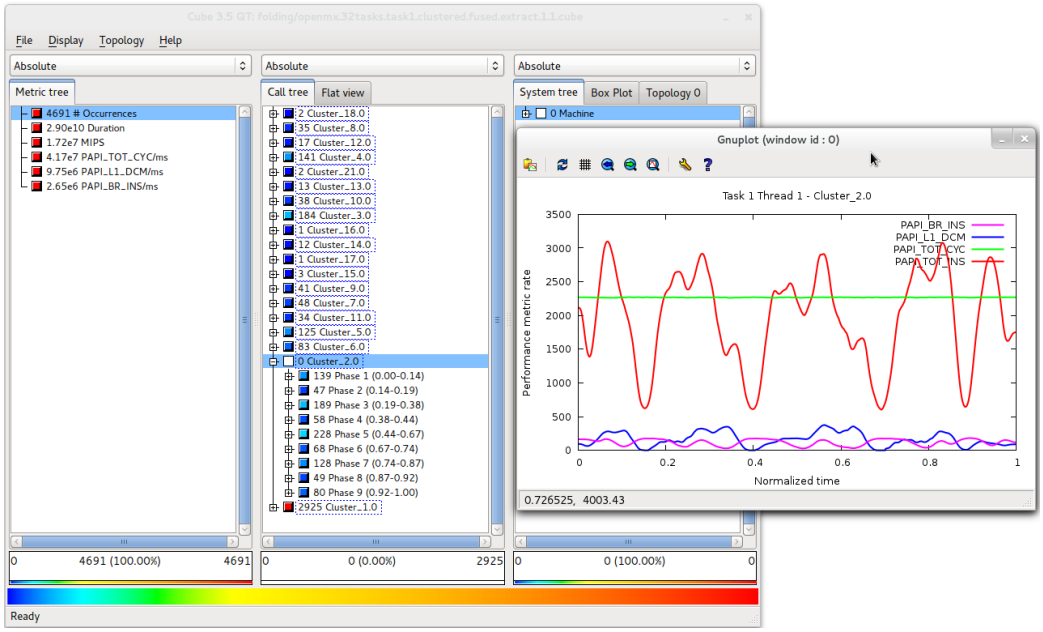

Figure 49: Folding analysis with Cube – Cluster 2 all metrics

3.1.2 Scalasca delay analysis of the CESM sea ice model

In this case study, we applied Scalasca to a trace experiment of the Community Earth System Model (CESM) [29] climate code, where the delay analysis illustrates problems in the domain decomposition and communication pattern. The measured configuration used a 1º dipole grid of the Earth and a Cartesian grid decomposition on 2048 MPI processes. This configuration suffers from severe load imbalances. The first of these imbalances is a result of the domain decomposition in the form of a uniform, high-resolution grid of the Earth. Since the sea ice model obviously only applies to ocean regions, processes assigned exclusively to land regions do not participate in computation or communication at all. Also, processes assigned to regions where land and ocean overlap have less workload than processes assigned to all-ocean regions. A second load imbalance exists between processes assigned to open ocean regions and those assigned to sea ice (i.e., polar) regions, which have a significantly higher workload than the others. As a result of these imbalances, many processes suffer late-sender wait states in the point-to-point nearest-neighbour MPI data exchange following the computation.

Figure 50 through Figure 55 visualize the performance metrics determined by Scalasca in the 64x32 2D process grid used by the sea ice simulation. To allow an easy comparison between the different performance metrics, the figures only show the topology (right-most) panel of the Cube result browser. Red colours denote high values, blue colours denote low values. On the top left, Figure 50 illustrates the distribution of late-sender waiting time across the computational grid. Essentially, all processes assigned to open ocean regions incur wait states. Note that processes assigned to land-only grid points do not participate in the data exchange at all, and therefore do not incur late-sender wait states.

The delay analysis now allows us to understand the original causes of the wait states and the wait-state propagation pattern. It pinpoints source code and process locations where excess computation time leads to wait states at subsequent synchronization points. Delay costs denote the total amount of waiting time caused by a particular source code / process location. As shown by the distribution of delay costs in Figure 51, the delays responsible for the late-sender wait states in the sea ice model are located on the border between sea ice and open ocean processes. Processes assigned to an area north of Japan particularly stand out: further investigation revealed that this is likely the result of the

complex topography around the Kamchatka peninsula, which makes the sea ice computations more expensive for this area. The delay analysis easily pinpointed this area as a hotspot of the simulation.

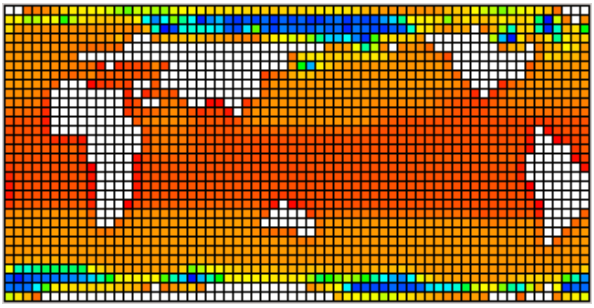

Figure 50: Total late-sender wait states

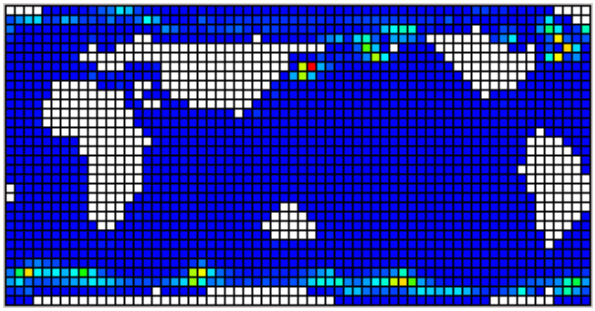

Figure 51: Delay costs

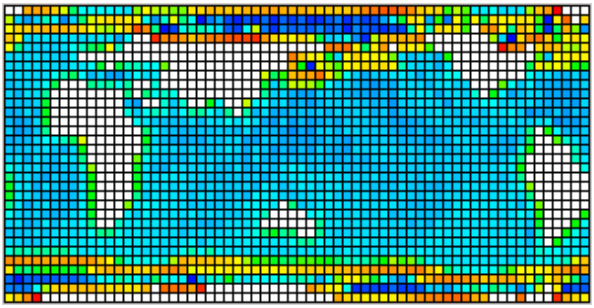

Figure 52: Direct late-sender wait states

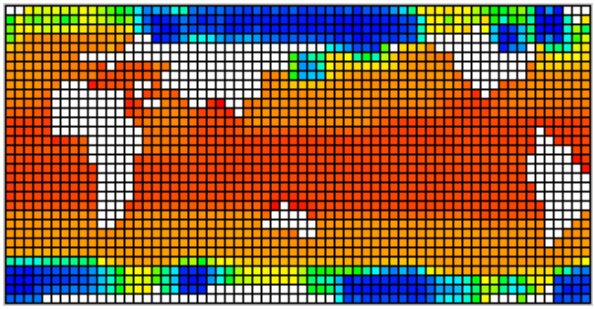

Figure 53: Indirect late-sender wait states

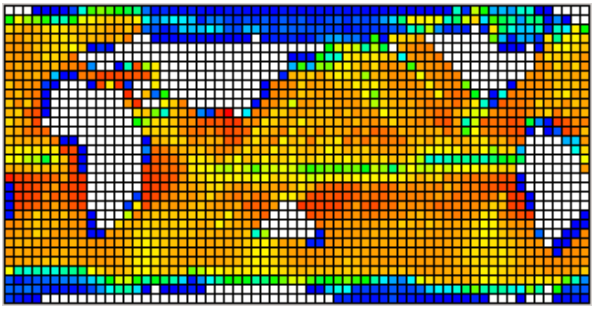

Figure 54: Propagating late-sender wait states

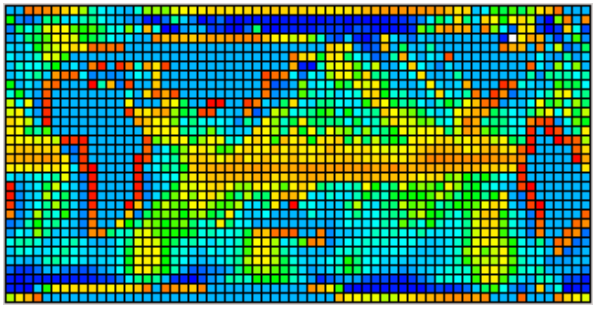

Figure 55: Terminal late-sender wait states

Because of the nearest neighbour communication pattern, the delays on the open sea/ice border only affect a small number of processes directly. However, the wait states they cause propagate to processes further away in subsequent iterations, so that they eventually affect the entire process grid. The classification of wait states into direct versus indirect and propagating versus terminal wait states performed by the delay analysis allows us to observe the propagation effect directly. First of all, the distribution of indirect wait states in Figure 53 confirms that wait states on most processes are indeed produced by propagation. Direct wait states, which are the wait states directly caused by excess computation, are only located on processes immediately surrounding the sea ice regions in the process grid, as can be seen in Figure 52. Propagating wait states are wait states which themselves delay subsequent communication operations, thereby causing more wait states on other processes. From the distribution of propagating wait states in the sea ice simulation shown in Figure 54, we can see that wait states propagate both southwards from the northern polar region as well as northwards from the southern polar region towards the equatorial region of the domain. Terminal wait states (Figure 55), which do not affect subsequent communications, are primarily located on processes assigned to the equatorial ocean regions and on shores, where the propagation chain ends. Hence, with the help of the delay analysis, we are able to gain detailed insight into the causes and propagation pattern of wait states.

It should be noted that the experiments were run at a larger scale than typically used in production runs. The limited scalability of the regular domain decomposition for the sea ice calculation is known, and is expected to produce significant imbalance for large-scale runs. However, these large-scale experiments magnify domain decomposition and communication issues early, and therefore provide useful performance insights. Currently, the developers investigate better load balancing schemes, in particular space-filling curves, to achieve better balance at large scale.

3.1.3 Vampir System Background View

Vampir was improved to select, display, and associate performance information in the scope of system nodes rather than individual processes. This new feature has been integrated into Vampir's "Counter Data Timeline" and "Performance Radar". It has been successfully applied on the Russian system "GraphIT" by visualizing HOPSA node level metrics as well as in current research of additional projects [18][30].

For the evaluation of the "System Background View" on the "GraphIT" system, the benchmark code HPL and a cannon algorithm implementation have been instrumented with Score-P. In addition to application and MPI events also specific HOPSA node level metrics were recorded.
This integration allows analyzing the effect of node level metrics on the application execution. In the HPL code visualization (Figure 56) one can see rising floating point operations (second timeline) resulting in a higher memory consumption per node (third timeline).

Another powerful approach is to record InfiniBand communication between nodes. This allows the identification of possible communication bottlenecks due to problems of the InfiniBand network. Therefor a MPI implementation of a cannon algorithm has been traced along with the measurement of InfiniBand performance data counter. Figure 57 shows the MPI communication next to the InfiniBand data metrics per node. The correlation between both data sets becomes clearly visible.

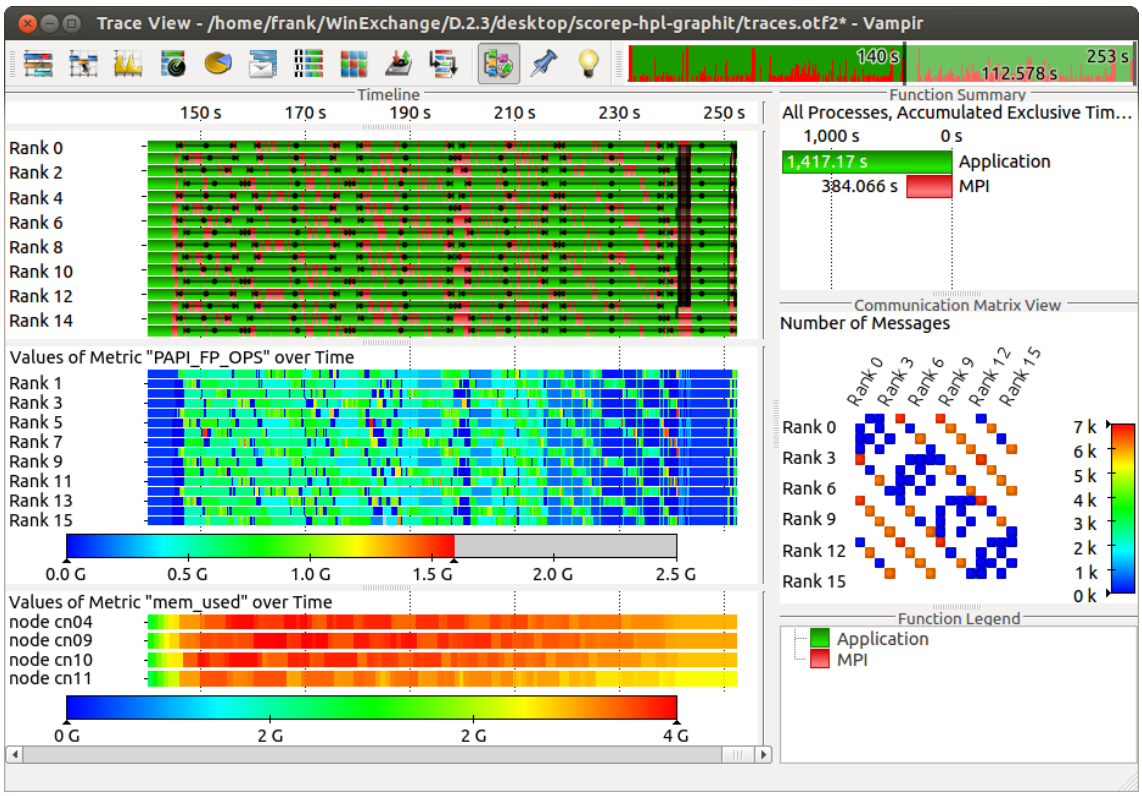

Figure 56: Vampir's Trace Visualization of the benchmark code HPL including the HOPSA node level metric "mem_used" (used memory) in the Performance Radar

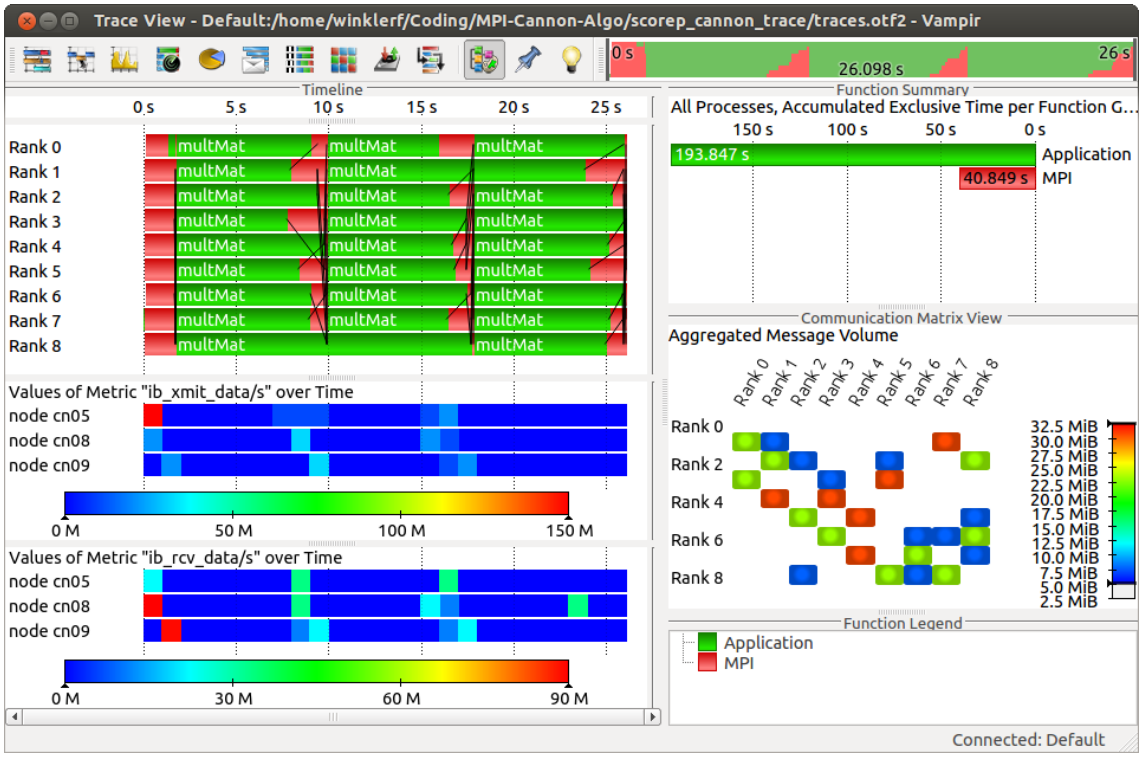

Figure 57: Vampir's Trace Visualization of a cannon algorithm showing the MPI communication next to the InfiniBand data counter

Vampir's "System Background View" is also used in the scope of current research projects. One example is the project CoolComputing [18], funded by the "Bundesministerium für Bildung und Forschung" via the research project CoolSilicon (BMBF 16N10186). Goal of this project is the analysis of the energy consumption of individual program phases of an application. In the scope of this project the energy consumption of the benchmark "122.tachyon" from SPEC MPI2007 [20] was measured in detail. By using the new "System Background View" it was possible to obtain the following results.

Obviously, the power consumption is correlated with the number of instructions per cycle, as shown in Figure 58. An unexpected result is depicted in Figure 59. Shortly before the sending of MPI messages starts the energy consumption of the application drops. This observation needs further investigation. Figure 60 shows the final *MPI_Waitall* call of the benchmark. An increased memory access is observed, resulting in higher energy consumption per node. This expected result is documented in [21].

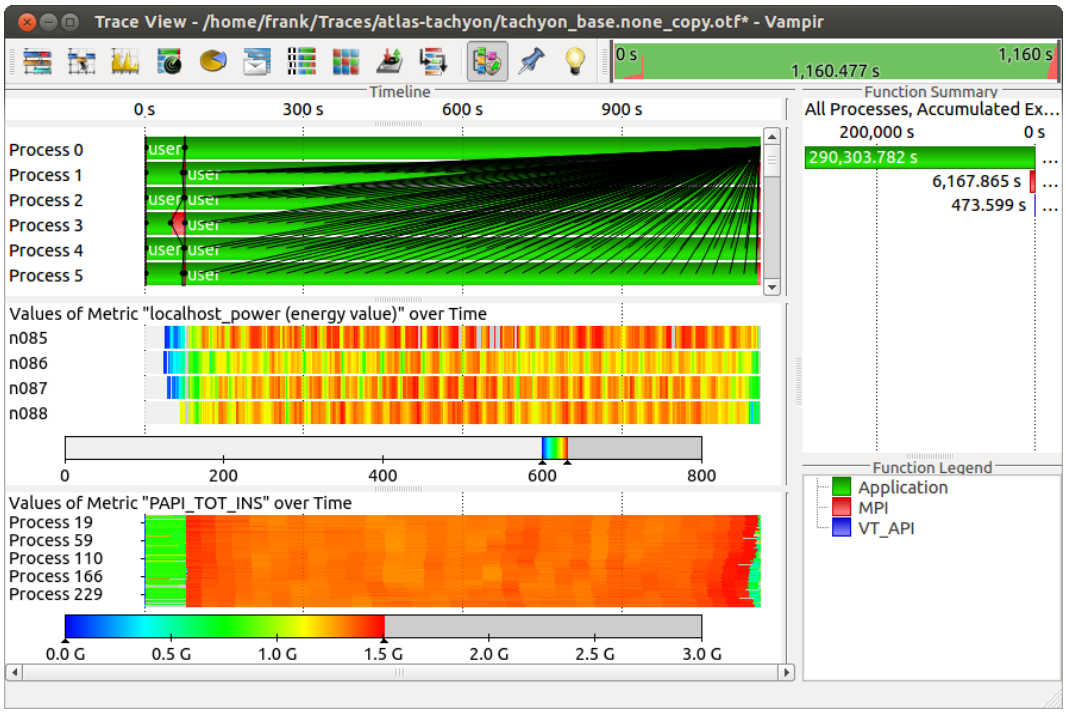

Figure 58: "122.tachyon" Benchmark: The power consumption per node correlates with the number of instructions per cycle

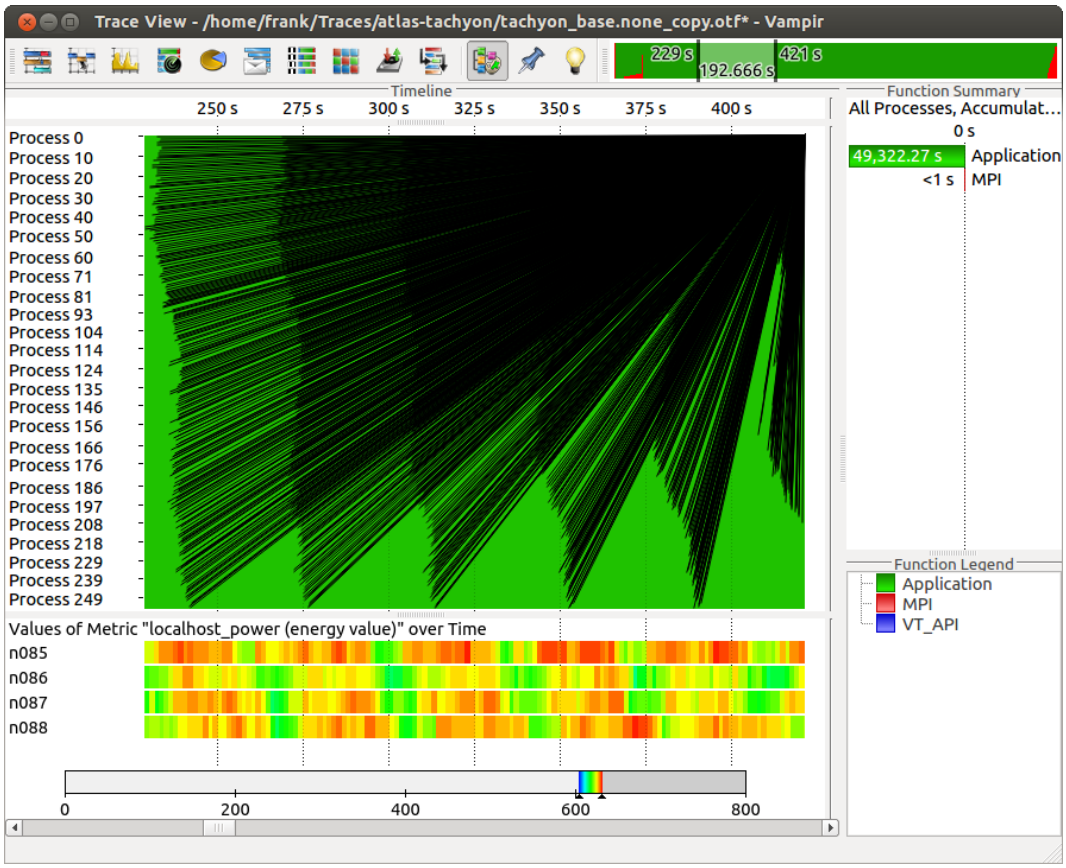

Figure 59: "122.tachyon" Benchmark: Shorty before the sending of MPI messages the energy consumption drops slightly

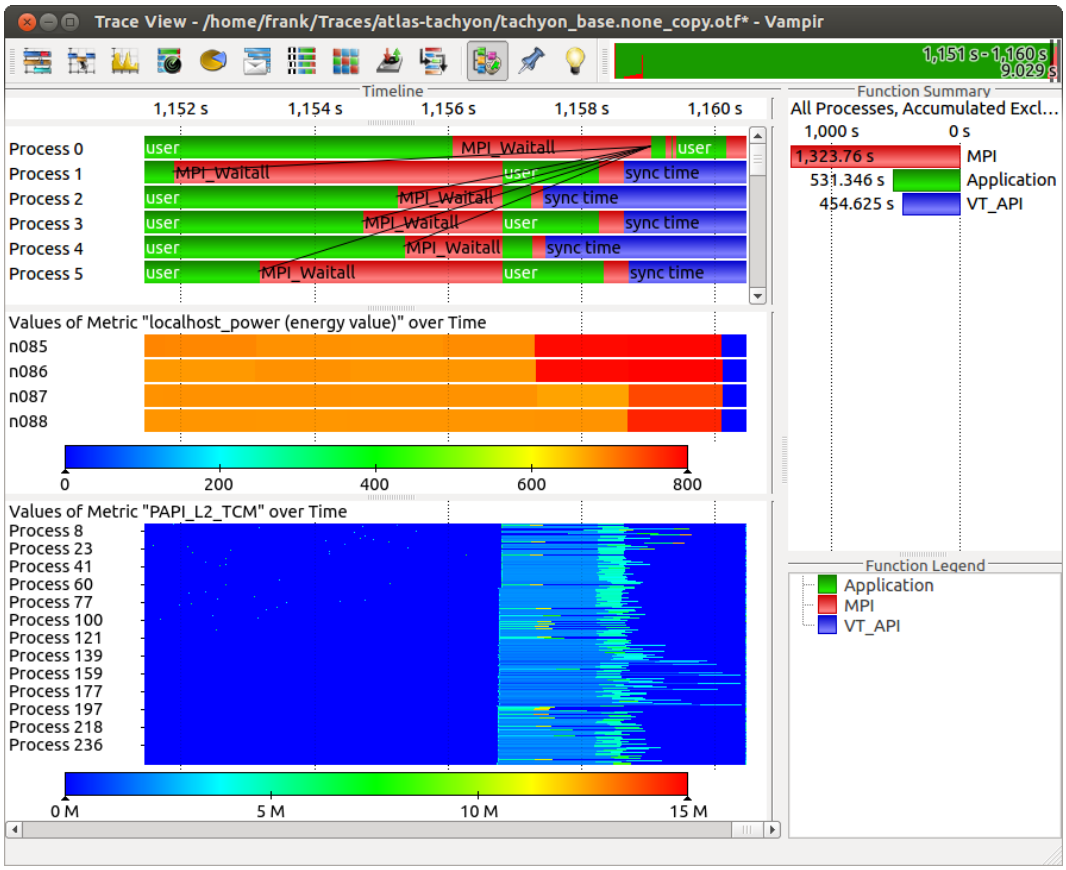

Figure 60: "122.tachyon" Benchmark: The power consumption per node correlated with the memory access

3.2 Scalability enhancements developed within HOPSA

3.2.1 BSC Tools

The scalability enhancements implemented in BSC tools within the framework of HOPSA targeted the three main components: Paraver, Dimemas and Extrae. As Paraver is an interactive tool, there is not a good mechanism to report the effective improvement on the full interactive analysis process, but both main computations (timelines and tables) have been parallelised for a multi-core with an efficiency around 80-95%. The higher efficiencies are obtained on the parallelization of the timeline, while the lower values correspond to the tables where the tool computes global statistics.

Dimemas scalability

Dimemas is a simulator that target what-if analysis. To support the exploration of potential scenarios it is important that the simulations are calculated quickly enough so the analyst is not tempted to avoid new simulations. One of the most useful functionalities of the simulator is its ability to generate a Paraver trace file as output allowing comparing and understanding different simulations and the original instrumented execution.

The improvement of the Dimemas scalability had two main targets: on one side to improve the simulation of applications with a large number of MPI ranks and on the other side eliminate the restrictions to generate the Paraver output trace caused by the huge number of file descriptors used by the old version.

To validate the results, we selected 4 traces of real codes running at medium-large scale. Figure 61 characterizes the different traces used with respect to its records. As a summary Zeus and Gromacs traces are larger than PEPC and Linpack but they correspond to runs with lower number of tasks.

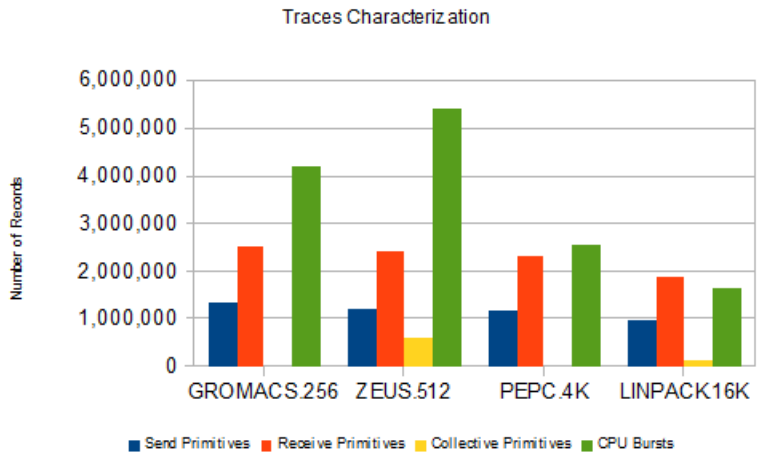

Figure 61: Profile of the traces used as reference

Figure 62 compares the execution time of the new version (v4) with the Dimemas distribution before HOPSA project (v3). For each Dimemas version we considered four scenarios: activating or not the generation of a Paraver output trace and with or without limiting the number of file descriptors to half of the application tasks. With Dimemas v3 it was not possible to generate an output trace if the number of file descriptors was limited.

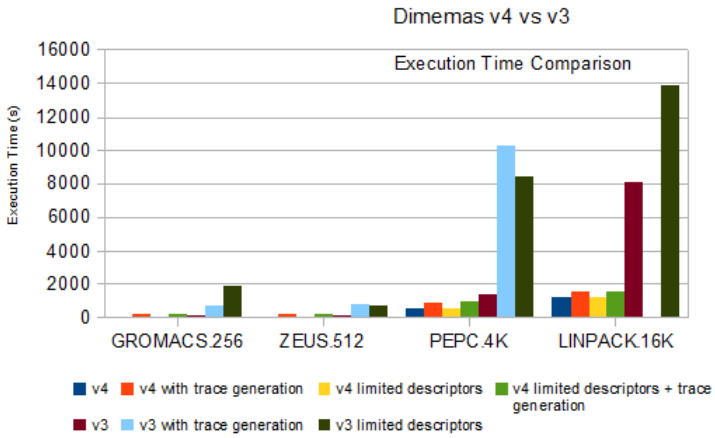

Figure 62: Dimemas execution time

The first observation is that there has been a huge improvement for the larger cases (4k and 16k tasks) Dimemas v3 execution time was very sensitive to the number of available file descriptors as well as to the generation of the Paraver output tracefile. Even worst, it was not possible to generate a Paraver output trace file for the 16k runs even with enough descriptors. These problems have been fixed with Dimemas v4 that presents a reasonably small variability on the four configurations. A final remark is that even the traces of Gromacs and Zeus were larger with respect to the number of records, the simulation required less time indicating the management of a large number of tasks is dominating over the number of simulated records.

Figure 63 plots the memory usage for the same Dimemas executions that is basically proportional to the scale of the application being simulated with small variability in the different versions and set-ups. In this case the new version slightly increased the memory usage.

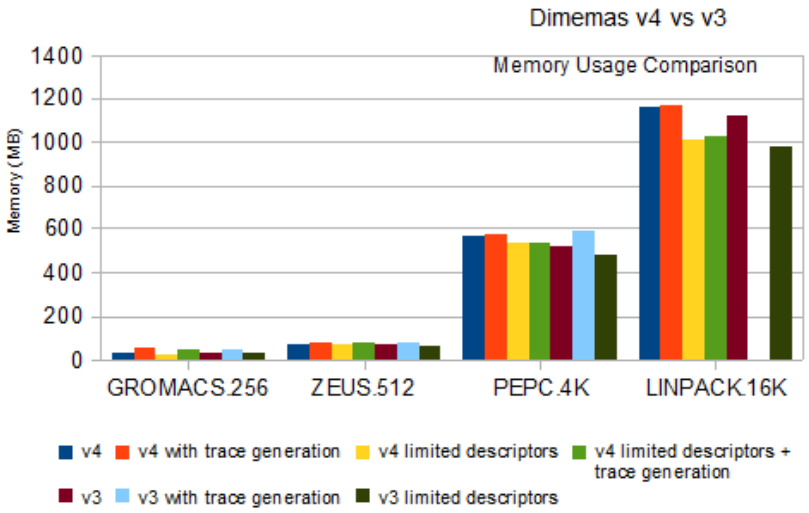

Figure 63: Dimemas memory usage

These results validate the benefits of the improvements implemented in Dimemas within HOPSA as well as identify potential future improvement with respect to the memory used and the search over the event queues and structure.

Instrumentation scalability through on-line analysis

The target to increase the scalability of the measurement has been tackled by reducing the volume of data captured. The mechanism is to incorporate the instrumentation phase algorithms used in the post-mortem analysis to detect the relevant regions for the analysis. Within HOPSA we have ported the pattern analysis module to work on-line using the instrumentation buffer [36].

To validate the impact and results of using this module, we applied it to the analysis of the PFLOTRAN code. The instrumentation of the full execution generates a trace very close to 5GB that is drastically reduced by one order of magnitude to 400MB when the run is instrumented activating the on-line pattern analysis. Although the reduction will depend on the number and variability of the iterations, in most of the cases where we applied this approach, a very similar magnitude of reduction was achieved.

For that reason the information we report on this deliverable is about the information included on the trace files obtained by this mechanism. Figure 64 provides the full execution global views. The top image displays the level of detail on the different regions, red meaning detailed information (like the standard instrumentation), dark blue profiled regions (few metrics accumulated per iteration) and black no information collected (the black areas in the middle of the execution correspond to the intervals where the application is stopped to do the on-line analysis). The second timeline uses a different color for each detected phase with a different iteration. The third timeline marks with different colors the iterations detected within the execution.

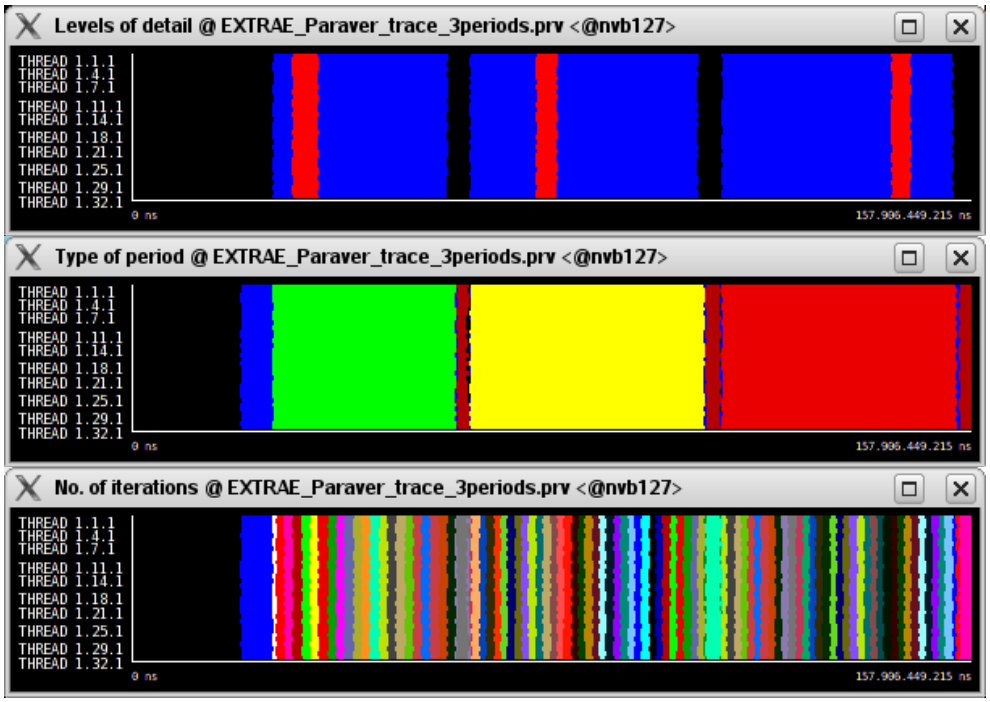

Figure 64: Full execution views

The detailed analysis is done within the two iterations per phase that are included in the trace file. Figure 65 shows the MPI calls for one of the regions instrumented in detail. The benefit of this mechanism is that it focuses the attention of the analyst on representative regions simplifying the study.

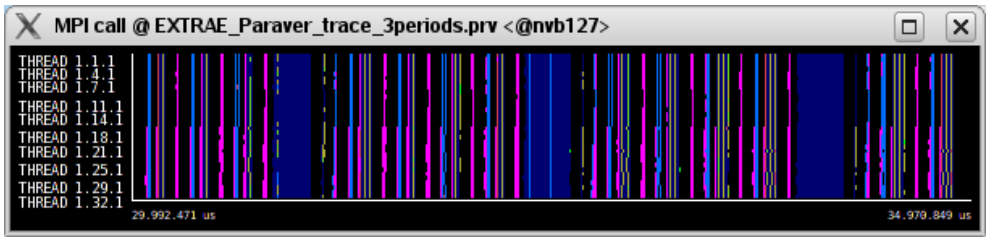

Figure 65: MPI calls on a detailed region

It may be interesting to understand what it is the difference between the different types of iterations detected by the on-line analysis. To simplify the comparison we will focus only on the first two types of iterations detected. Figure 66 shows the code references only one iteration of each of these phases. We can see that the iterations are very similar except for the number of the internal iterations colored as the sequence dark brown – pink – green with red – light brown that happened three times on the first type of iteration and only two times on the second type. Based on this difference, the analyst can decide if it is worth to look at the details of the second phase or he can extrapolate the analysis results of the first phase to the second one. In case the analyst does not want to differentiate small variations in the structure, there is a parameter that will suggest the tool to consider such variations as part of the variability of the same type of iteration.

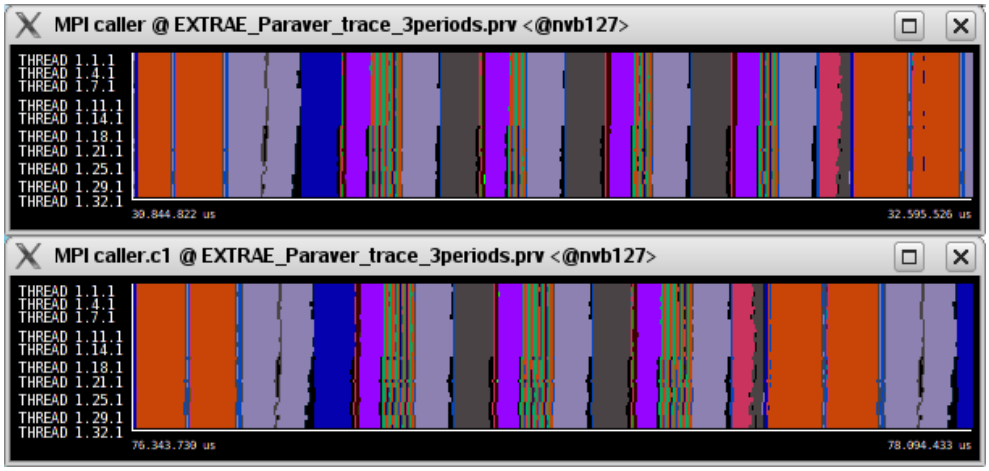

Figure 66: Comparison of the iterations' structure

3.2.2 ThreadSpotter MPI clustering

Overview

The purpose of MPI rank clustering is to reduce the number of reports and analyses that the user has to view after sampling a distributed job. The idea is that many ranks will show the same behavior and it would only be necessary to look at a few reports to gain sufficient knowledge of all interesting information, provided that the selected reports are well chosen. Interesting changes may be visible in programs having just one computation mode due to load imbalance or topology, but it is expected that the real benefit will be for MPMD programs (multiple programs / multiple data), where the MPI nodes assume different logical roles and perform different computations. Despite the acronym, MPMD could be taken on a logical level, where the binary is actually the same on all nodes and where the rank number lets each instance of the program figure out its role.

Clustering algorithm

The challenge was to find a representative set of data (“Condensate”) from each rank that is cheap to extract and that captures relevant differences. The data also needs to work as input to a grouping algorithm and the resulting grouping should resemble the grouping that one would subjectively make oneself based on the appearance of the final reports.

The resulting Condensate was finally selected to be based on per-PC reuse histograms, and histograms are compared using a variation of EMD (Earth mover’s distance). The condensate files from each rank contain ~100 values each and are easy to move about.

Clustering is based on PAM (Partitioning Around Medoids). The goodness of the clustering and the optimal number of clusters is estimated using the Silhouette metric.

General workflow

Sample files are obtained from all ranks. These are post-processed in parallel to extract the Condensate. The condensate files from all ranks are used as input to the clustering algorithm, which subsequently identifies the clusters and one representative rank from each cluster. The reporting tool then prepares reports for the representative ranks. The user can then view the representative reports.

Validation

During a transition period and subsequently for regression testing, a tool is provided that can be run offline on either a set of sample files or a set of condensate files and produce a clustering information file and then proceed to generate the required report files.

It can also compare the results from a clustering based on a condensate with a clustering based on high level statistics. The goal is to validate clustering qualities based on the correlation of these two

results. The correlation is presented in a graphical tool, in which we can also modify parameters to the clustering algorithms.

Example

We present the GROMACS application as an example. **GROMACS** [26] is a molecular modeling tool, and it can run in different modes. We use the MPI enabled mode and run the model on 32 nodes of a Cray XE6 cluster. The nodes transparently organize themselves into two roles, but normally without the user having any say in exactly how that is done. The application is sampled through normal means (by launching a sampler in each node, and letting each such instance invoke the application). This creates 32 different sample files and the tools shows a strong indication for the presence of two different roles.

As an illustration of this, first consider the internal validation tool, see Figure 67.

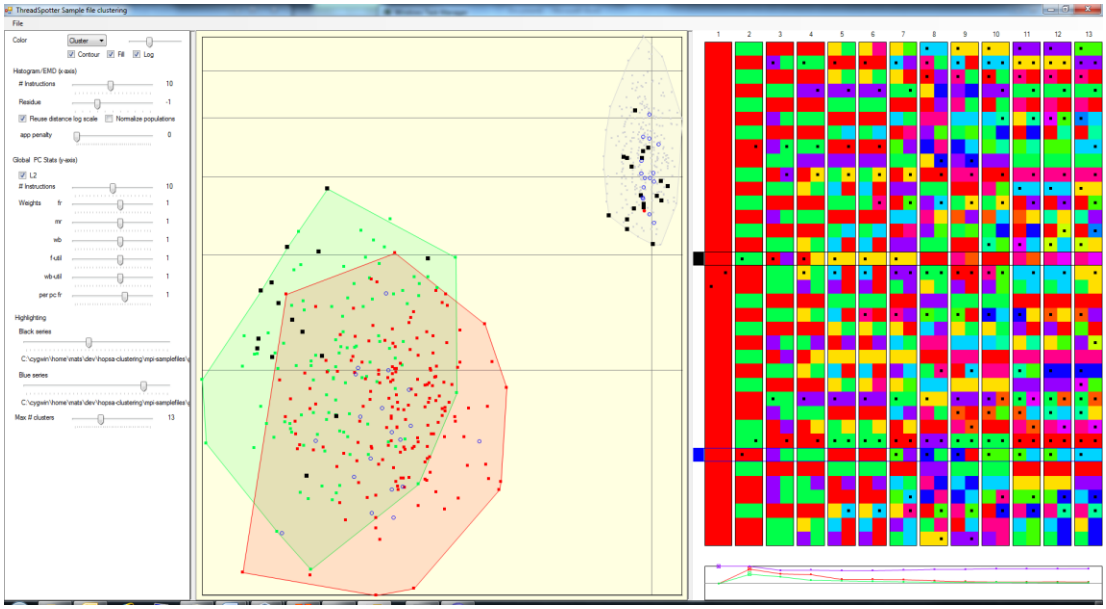

Figure 67: ThreadSpotter: Internal validation tool

Figure 67 shows (to the left) a panel of parameters that we can modify, (center) a diagram of how clusters are formed and distances between different ranks, and (right) a visualization of which ranks are considered to belong to the same cluster. The second column (with alternating red and green rectangles) represents the optimal clustering. Selecting two resulting clusters produces the maximum Silhouette metric (shown at bottom to the right). Other selections of number of clusters will produce an inferior Silhouette. A good Silhouette metric indicates a low within-cluster distance, and a good separation between clusters. This is also evident in the center image, where the red and green clouds represent low intra-cluster distances, while the gray cloud represents the resulting between-cluster distances (such as differences between members of the red and green cluster). These have a clear separation from the red and green intra-cluster distances. Clearly this is a good separation.

Each of the so identified clusters will be represented by one representative rank, and the corresponding sample file will be used to create reports as seen in Figure 68 and 69.

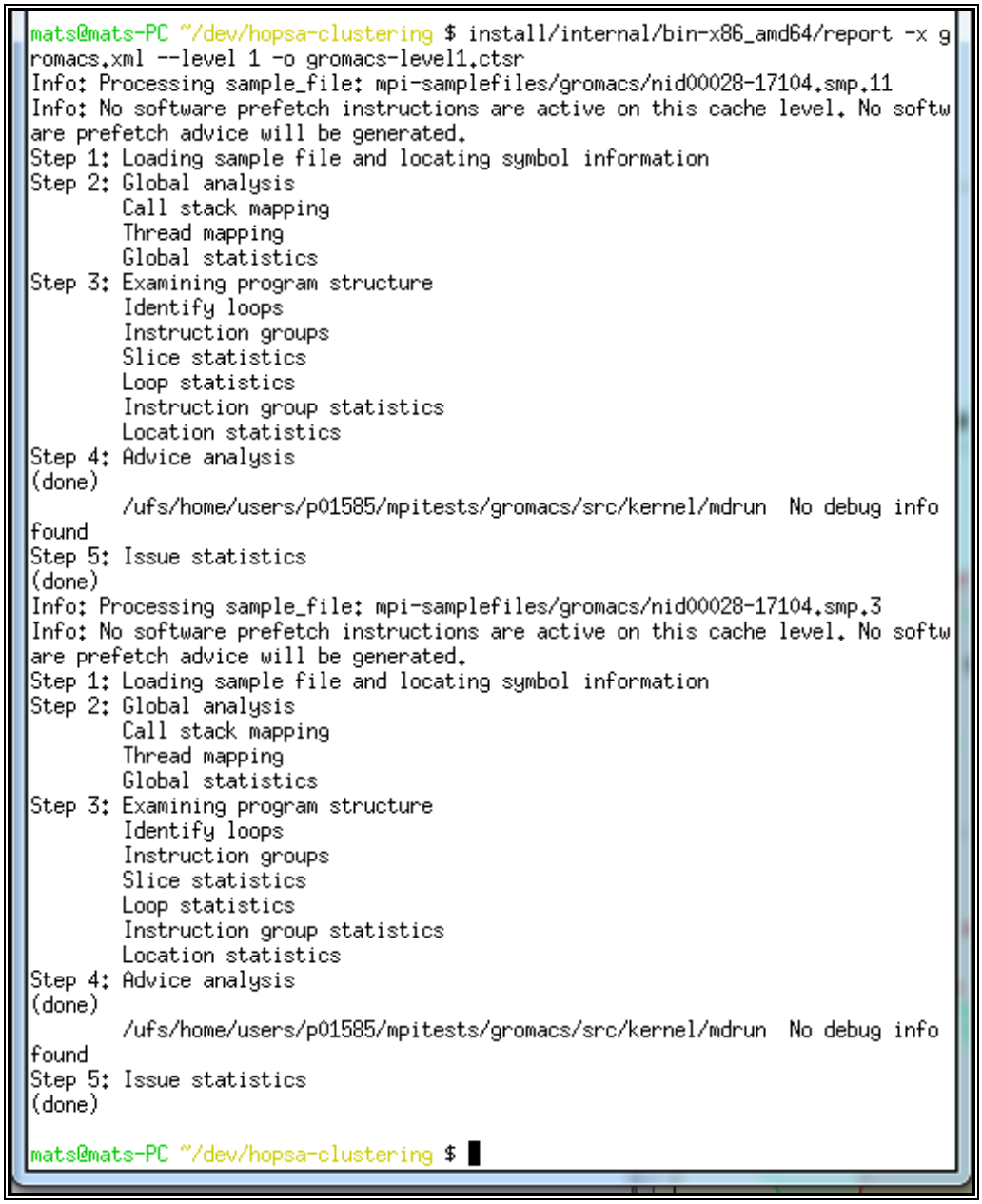

Figure 68: ThreadSpotter: A session where the clustering algorithm has identified two distinct clusters and proceeded to create reports for a representative member from each of the identified clusters.

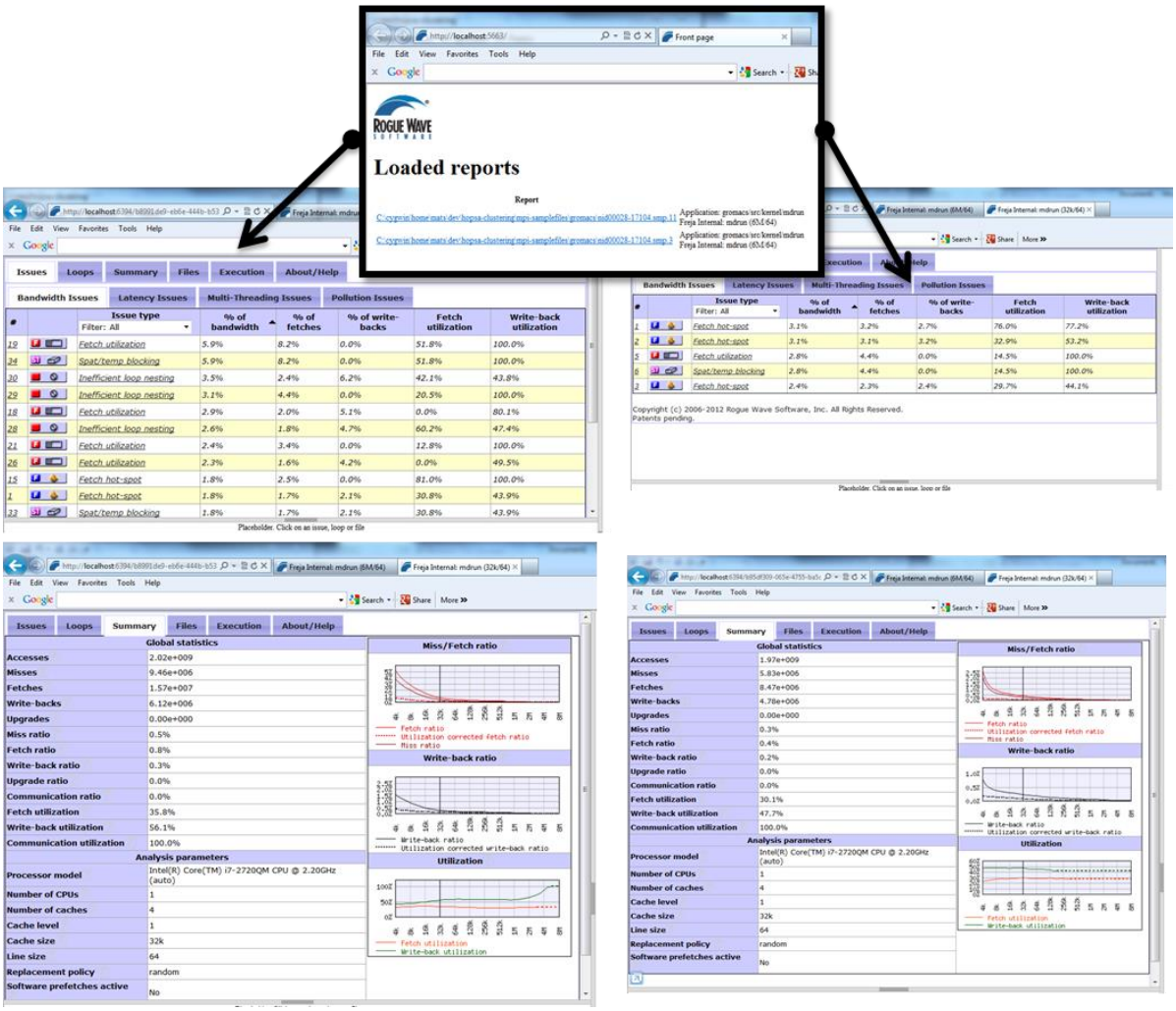

Figure 69: ThreadSpotter: The resulting reports from the previous session (Figure 68). The user can navigate to the two representative reports. The picture shows that advice and statistics differ significantly between the two reports.

3.2.3 Vampir scalability experiment

Within the HOPSA project the scalability of Vampir analysis has been further improved. Based on the Community Earth System Model (CESM) [29] climate code running on the Titan machine at Oak Ridge National Laboratory (ORNL) a trace of about 16 gigabytes (compressed) has been created to investigate Vampir's parallel analysis engine with various configurations. Titan is a supercomputing system with a hybrid architecture based on 16-core AMD Opteron CPU's and NVIDIA Tesla K20 GPU accelerators with a theoretical peak performance of more than 20 petaflops. The CESM code has been traced for 1280 MPI processes. Figure 70 shows a screendump of the CESM code visualization in Vampir.

In order to validate the scalability of Vampir the trace file has been loaded with different numbers of analysis processes. For this purpose, the time was measured until the Vampir displays "Master Timeline" and "Function Summary" were completely loaded. In Figure 71 the loading times depending on different numbers of analysis processes are shown. Loading the trace file with 32 analysis processes was not possible due to insufficient memory (32 gigabytes RAM per node). The most efficient configuration is 512 analysis processes (47 seconds loading time), this allows very smooth working, only the "Master Timeline" takes a few seconds for an update by zooming. A doubling of the analysis processes to 1024 or 1280 enables a slight decrease in loading time but is not as efficient as the configuration of 512 processes because there is no enhancement of the interaction times. An

explanation of this behavior is the data layout in the trace file. Using 1280 analysis processes each process needs to read only one file. This results in a faster loading time compared to the case with 512 analysis processes, where each process needs to load at least two files. However, the processing of the trace data for the visualization such as zooming does not provide any improvement of the interaction time. An explanation could be the small data size of each process which is between 11 and 76 megabytes and therefore does not make any difference compared to the case with 512 analysis processes. In the shown example of the analysis of the CESM code working with 512 analysis processes is the most efficient solution. For the analysis of larger application runs a higher number of analysis processes might be required.

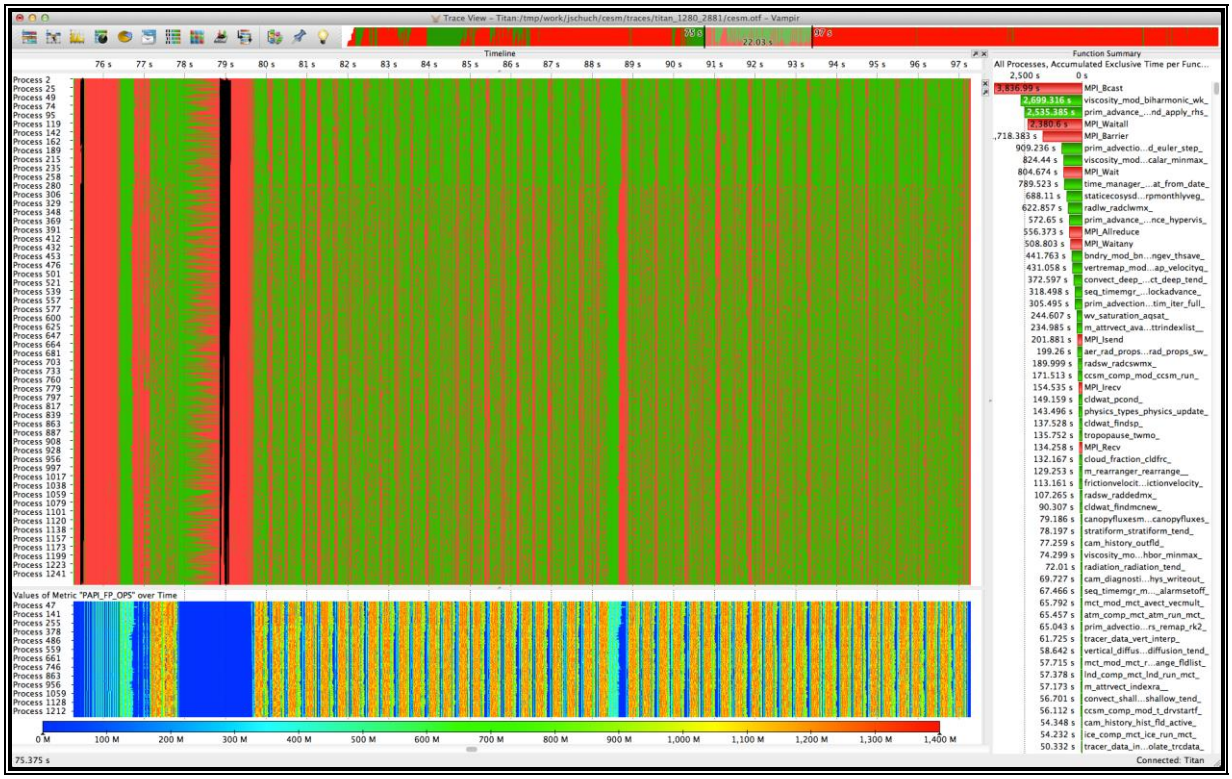

Figure 70: Vampir's Visualization of the CESM code running on 1280 processes

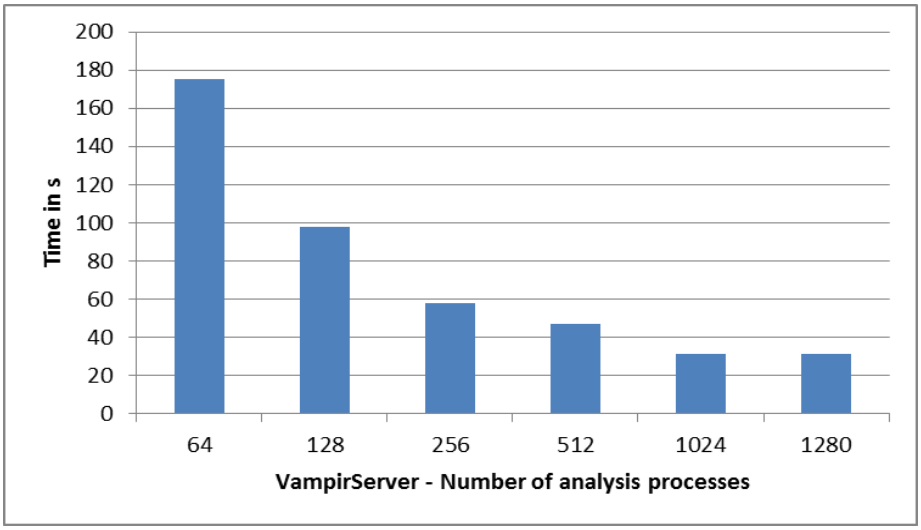

Figure 71: Vampir's loading time of the CESM trace with different numbers of analysis processes

3.2.4 Scalasca BlueGene/Q scaling experiments

To test the scalability of the various Scalasca measurement and visualization components, various scaling experiments have been done with the NAS parallel benchmark BT-MZ on Juqueen, a large 28-rack IBM BlueGene/Q system at JSC. Each rack provides 1024 IBM 1.6 GHz PowerPC A2 processors. Each processor features 16 cores and each core is four-way SMT. So, the complete system has 458,752 cores allowing to execute a total of 1,835,008 concurrent HW threads.

In summer 2012, we are able to successfully measure and analyse a NAS BT-MZ execution with 524,288 threads using 8 racks. Figure 72 shows the result of the Scalasca trace analysis for this experiment. It shows the distribution of the execution time of the routine *z_solve* over all threads. To facilitate the visualization of such large thread numbers, Scalasca uses topology displays. In Figure 72, the 3D mapping of the 6D hardware topology of the BlueGene/Q is used (consisting of the 5D network topology (ABCDE) and the thread id (T)). The mapping of the six dimensions to the 3D visualization dimensions (xyz) can be specified by the user (see bottom of the topology display on the right). By dragging the topology dimensions (i.e., the dark-gray boxes) with the mouse into the three x, y, and z rectangles, the topology can be easily rearranged.

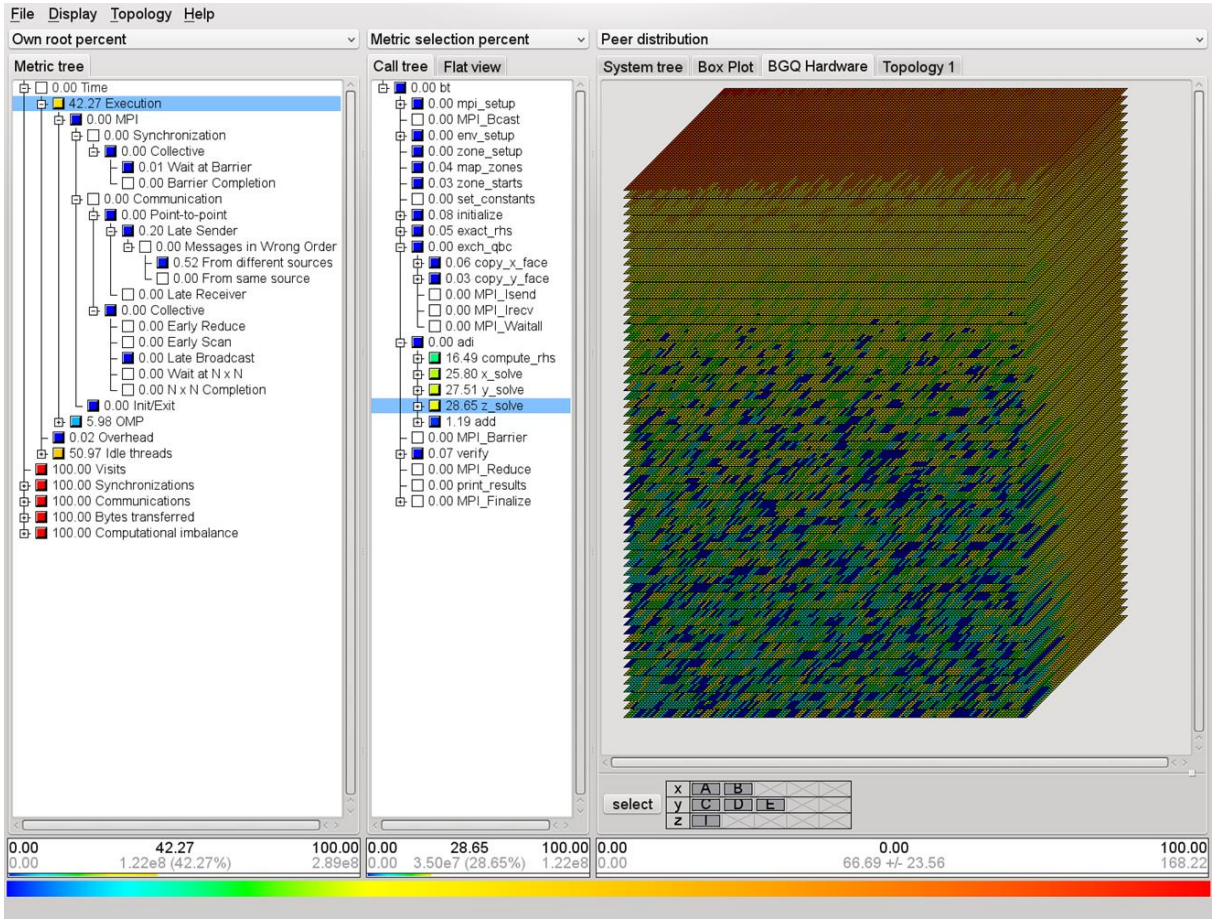

Figure 72: Scalasca trace analysis of NAS BT-MZ executed with 524,288 threads

Finally, in November 2012, we were able to repeat the experiment on 16 racks resulting in 1,048,576 threads. Figure 73 shows the distribution of execution time of the OMP parallelized loop inside the routine *z_solve*. This time, the topology has been arranged in a way that the MPI ranks run

horizontally in multiple rows, while the values for the threads are arranged vertically in each row. One can easily see that the work distribution of the loop is not perfect: higher numbered threadshave less to do (low yellow values compared to high red values).

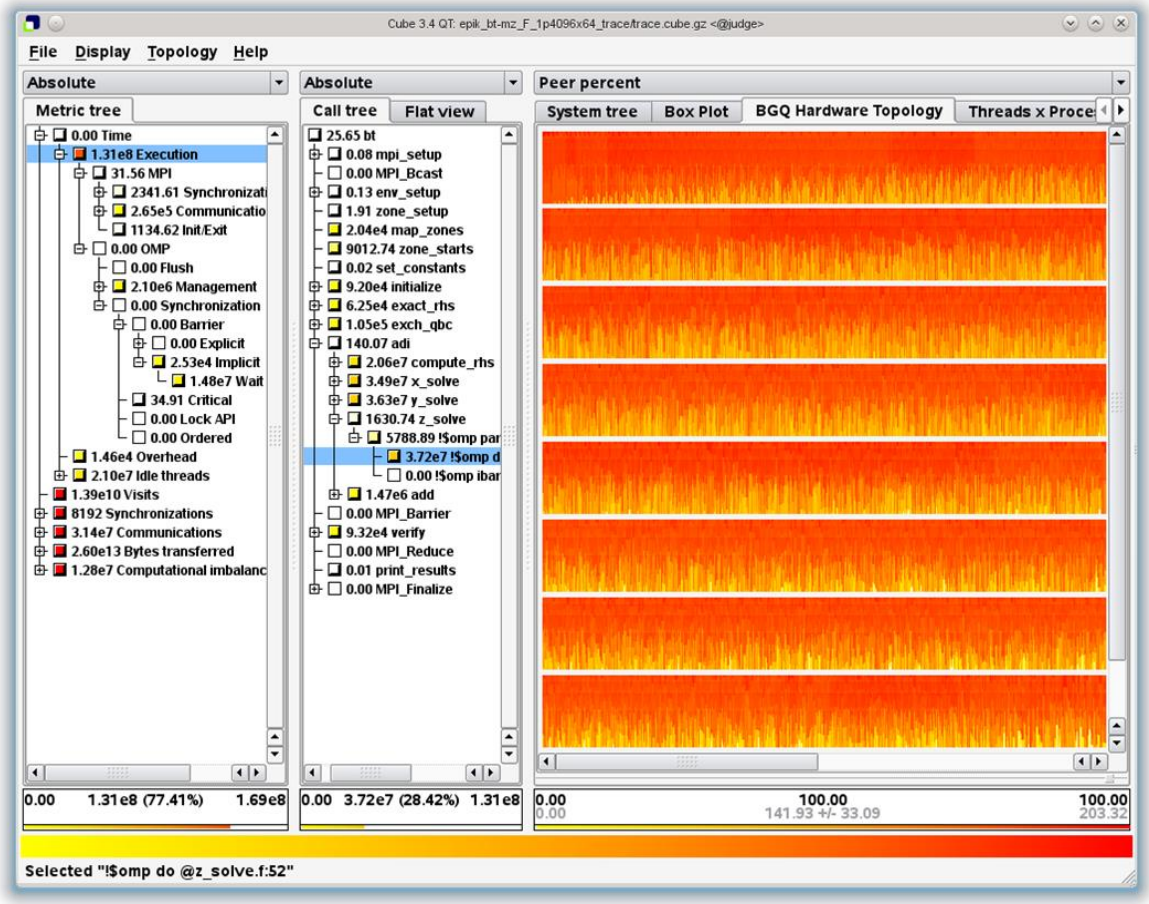

Figure 73: Scalasca trace analysis of NAS BT-MZ executed with 1,048,576 threads

To our knowledge, this is world-wide the first successful measurement and analysis of more than one million concurrent threads.

4. Conclusions

This document provides an overview about the results of the EU part of the HOPSA project. This includes an introduction of the individual tools and showcases a number of relevant benchmarks and applications from other EU FP7 projects. Furthermore, it demonstrates the extensions designed in the HOPSA project and validates the tools together with selected relevant application scenarios.

The report is intended to draw the attention of users as well as developers of performance analysis tools.

For new users this report gives a broad overview about the potential of dedicated performance analysis tools and many different aspects addressed by each of them. Furthermore, they will be able to figure out which tool or combination of tools will suit their specific requirements best.

Experienced tool users learn about the advancements of the tools including new features and scalability as well as interoperability. They will profit from the extensions through the HOPSA project in their daily work.

For readers involved in developing performance analysis tools this report demonstrates the state-of-the-art and the feature enhancements in the tools. Particular highlights are the newly added interoperability features between the tools, which are not only present in experimental version but will become part of the official release versions of all tools.

Overall, this report presents the successful EU part of the HOPSA project from the perspective of practical use cases with real-world applications.

5. Bibliography

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

[2] M. Geimer, F. Wolf, B. J. N. Wylie, B. Mohr: A scalable tool architecture for diagnosing wait states in massively parallel applications. *Parallel Computing*, 35(7):375-388, July 2009.

[3] B. J. N. Wylie, M. Geimer, F. Wolf: Performance measurement and analysis of large-scale parallel applications on leadership computing systems. *Scientific Programming*, 16(2-3):167-181, 2008, Special Issue Large-Scale Programming Tools and Environments.

[4] E. Hagersten, M. Nilsson and M. Vesterlund, Improving Cache Utilization Using Acumem VPE, *Tools for High-Performance Computing 2008*, III, 115-135, DOI: 10.1007/978-3-540-68564-7_8

[5] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. Müller and W.E. Nagel, “The Vampir Performance Analysis Tool-Set”, *Tools for High Performance Computing*, pp 139-155, Springer Verlag, 2008.

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

[7] J. Labarta, Trace-Based Tools, *Performance Tuning of Scientific Applications*, Edited by D. H. Bailey, R. F. Lucas and S. W. Williams, pp. 87–122, 2011.

[8] A. Snavely, X. Gao, C. Lee, L. Carrington, N. Wolter, J. Labarta, J. Giménez, P. Jones, *Performance Modeling of HPC Applications*, Proceedings ParCo 2003.

[9] V. Pillet et al.: PARAVER: A Tool to Visualize and Analyze Parallel Code, in: 18th World OCCAM and Transputer User Group Technical Meeting, April 1995.

[10] T-Platforms, Moscow, Russia. Clustrx HPC Software. http://www.t-platforms.com/products/software/clustrxproductfamily.html, last accessed September 2012.

[11] A.V. Adinets, P.A. Bryzgalov, Vad.V. Voevodin, S.A. Zhumatiy, D.A. Nikitenko. *About an approach to monitoring, analysis and visualization of jobs on cluster system* (In Russian). In: Numerical Methods and Programming, vol. 12, pp. 90–93, 2011.

[12] Bernd Mohr, Vladimir Voevodin, Judit Giménez, Erik Hagersten, Andreas Knüpfer, Dmitry A. Nikitenko, Mats Nilsson, Harald Servat, Aamer Shah, Frank Winkler, Felix Wolf, and Ilya Zhukov: *The HOPSA Workflow and Tools.* In: Proceedings of the 6th International Parallel Tools Workshop, Stuttgart, September 2012, Springer. To appear.

[13] D. Komatitsch, J. Labarta, D. Michéa, *A simulation of seismic wave propagation at high resolution in the inner core of the Earth on 2166 processors of MareNostrum*, in: J. Palma, P. Amestoy, M. Daydé, M. Mattoso, J. Lopes (Eds.), High Performance Computing for Computational Science – VECPAR 2008, Lecture Notes in Computer Science, vol. 5336, Springer, pp. 364–377, 2008.

[14] M. Garcia et al. LeWI: *A Runtime Balancing Algorithm for Nested Parallelism*, ICPP09.

[15] M. Migliore, C. Cannia, W.W Lytton, Henry Markram, and M. L. Hines: *Parallel Network Simulations with NEURON*, J. Comput. Neurosci. 21:110-119, 2006.

[16] S. Markidis, G. Lapenta und R. Uddin: *Multi-scale simulations of plasma with iPIC3D,* Mathematics and Computers in Simulation, Bd. 80, Nr. 7, pp. 1509–1519, 2010.

[17] G. Lapenta, J. U. Brackbill und P. Ricci: *Kinetic approach to microscopic-macroscopic coupling in space and laboratory plasmas*, Physics of Plasmas, Bd. 13, Nr. 5, pp. 055904-055912, 2006.

[18] Project Cool-Silicon, http://www.cool-silicon.de/projects/coolcomputing-technologien-fur-energieeffiziente-computing-plattformen/

[19] Standard Performance Evaluation Corporation, http://www.spec.org

[20] SPEC MPI Benchmarks, http://www.spec.org/mpi2007/

[21] D. Molka, D. Hackenberg, R. Schöne, M.S. Müller, *Characterizing the energy consumption of data transfers and arithmetic operations on x86−64 processors*, Green Computing Conference, 2010 International, pp. 123-133.

[22] A. Alvarez, M. E. Innocenti, F. Affinito, D. Henaux, G. Staffelbach, H. Merx, F. J. Delalondre, *Applications analysis and code division*, Confidential DEEP project deliverable D8.1

[23] S. Requena, B. Videau, A. Delorme. M. Culpo, R. Halver, S. Mohanty, D. Broemmel, J. Meincke, V. Moureau, M. Allalen, X. Saez, *Preliminary report of progress about the porting of the full-scale scientific applications*, Public Mont-Blanc project deliverable D4.1, see http://www.montblanc-project.eu/sites/default/files/d4.1_preliminary_report_of_progress_about...v1_reduced_.pdf

[24] J. Gracia, L. Arnold, C. Bekas, A. Curioni, D. Brommel, D. Komatitsch, M.-H. Nguyen, R. Reues-Castro, E. Quintana-Orti, C. Niethammer, V. Szeremi, *Optimized Porting of Applications and Kernels*, Public TEXT project deliverable D5.3, see http://www.project-text.eu/sites/default/files/20110301_TEXT_D5.3.pdf

[25] SPECFEM3D application, http://www.geodynamics.org/cig/software

[26] GROMACS application, http://www.gromacs.org/About_Gromacs

[27] Nek5000 application, http://nek5000.mcs.anl.gov/index.php/Main_Page

[28] ECS G8 Project, http://www.vi-hps.org/projects/ecs/

[29] Community Earth System Model, http://www.cesm.ucar.edu/

[30] EU ITEA2 project H4H (http://www.h4h-itea2.org/)

[31] H. Servat, G. Llort, J. Gimenez, J. Labarta: *Detailed Performance Analysis Using Coarse Grain Sampling*. Euro-Par Workshops 2009: 185-198

[32] H. Servat, G. Llort, J. Gimenez, K. A. Huck, J. Labarta: *Unveiling Internal Evolution of Parallel Application Computation Phases*. ICPP 2011: 155-164

[33] IBM, CPI-stack model, https://www.ibm.com/developerworks/library/pa-cpipower1

[34] IBM, CPI-stack model, https://www.ibm.com/developerworks/library/pa-cpipower2

[35] OpenMX application, http://www.openmx-square.org

[36] G. Llort,, M. Casas-Guix, J. Servat,k. Huck, J. Gimenez, J. Labarta,, *Trace Spectral Analysis toward Dynamic Levels of Detail*. 17th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2011, Tainan, Taiwan, pp. 332 – 339, 2011.

[37] EU FP7 Project DEEP, http://www.deep-project.eu

[38] EU FP7 Project Mont-Blanc, http://www.montblanc-project.eu

[39] EU FP7 Project CRESTA, http://www.cresta-project.eu

[40] EU FP7 Project TEXT, http://www.project-text.eu

