Assessing Measurement and Analysis Performance and Scalability of Scalasca 2.0

Ilya Zhukov and Brian J.N. Wylie

Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany

Abstract. The Scalasca toolset was developed to provide highly scalable performance measurement and analysis of scientific applications on current HPC platforms, including leadership systems such as IBM Blue-Gene/Q and more traditional Linux clusters. Its primary focus is support for C/C++/Fortran applications using MPI and OpenMP, and mixed-mode combinations thereof, offering detailed call-path profiles for each process and thread produced by runtime summarization or augmented with wait-state analysis of event traces. A new generation of Scalasca (2.0) uses the community-developed infrastructure comprising of Score-P and associated components, while continuing to provide the previous functionality. By comparing the new version of Scalasca with its predecessor, using the applications from the NPB3.3-MZ-MPI benchmark suite, we validate core functionality and assess overheads and scalability. Although adequate for general use, various aspects are identified for further improvement, particularly for larger scales.

Keywords: parallel performance measurement/analysis, scalability, MPI+OpenMP.

1 Introduction

Various studies have compared the usability and overheads of different performance tools for MPI applications on IBM BlueGene systems [1,7] and Linux clusters [5], however, we are particularly interested in comparing two generations of the Scalasca toolset [3] and its implementations on these two quite different classes of HPC computer system. We also focus on measuring and analysing the performance of combined OpenMP with MPI in recognition of the current predominance of this parallelization paradigm in the most scalable applications.

2 Scalasca and Score-P

Scalasca analyses of the execution performance of parallel applications consist of detailed call-path profiles for each process and thread which allow both quantification and localization of performance issues. Metric severity differences between processes and threads, which may indicate imbalance, and the program call-paths where these occur, can be readily identified and compared, or refined via hierarchical exploration. A wide range of valuable metrics can be determined independently for each process and thread during measurement via runtime summarization. Additional metrics indicating more specific parallel execution inefficiencies, requiring correlation of events on different processes that is prohibitive during measurement, can be determined via a parallel replay analysis of event traces. Locations of particularly severe instances of such issues found in the event traces can also be used to direct interactive trace visualization tools such as Vampir [4] to present them for detailed investigation.

While this usage has been supported for some years by Scalasca, community-developed software components promise closer integration with other tools and a larger basis for extending and improving functionality. For this purpose a number of Scalasca components have been re-developed with and for the community (the OPARI source instrumenter and CUBE analysis report utilities and GUI), along with a new OTF2 event trace format, however, the main new development is the Score-P instrumentation and measurement infrastructure [6]. The latter provides a functional replacement for Scalasca's original infrastructure (known as EPIK), generating summary analysis reports in CUBE format and/or event traces in OTF2 format for Scalasca's parallel trace analyzer (SCOUT).

Existing users of the Scalasca toolset are therefore able to continue to use the familiar commands to instrument their parallel application programs (SKIN), launch the instrumented executables under the control of the measurement configuration and analysis nexus (SCAN), and post-process and examine the resulting summary and trace analysis reports (SQUARE).

The new Scalasca instrumenter essentially only offers limited backwards compatability, since the Score-P instrumenter itself provides all of the functionality and can be used directly. Each instrumenter operates in the same fashion, using OPARI and other source instrumenters as appropriate, though command-line switches and manual instrumentation API have incompatible syntax.

The Scalasca measurement execution configuration and analysis nexus should also appear familiar in its operation. As before it verifies and sets the execution configuration for a measurement experiment, validates the resulting measurement artifacts and where appropriate automatically initiates parallel trace analysis using the same configuration of compute nodes, processes and threads. The most visible difference is that experiment archive directories are distinguished with the `scorep_` prefix (rather than `epik_`). While command-line configuration options are essentially unchanged for basic operation, Score-P environment variables must be used for detailed configuration and although many are similar to those for EPIK measurements there are also notable differences. For example, whereas EPIK uses separate memory buffers for call-path profile and event trace data on each thread along with per-process definition buffers, Score-P uses a memory buffer pool on each process for all of its internal measurement management and collection. As well as being more convenient, the latter pool can also be more memory efficient. Similarly, a file with a list of routines to filter during measurement can be specified in the same way on the SCAN command-line (and also for testing with SQUARE), however, its Score-P syntax is different.

For tracing experiments, event trace files with associated definition files are written directly into the archive directory as before, but the OTF2 library is used by the Score-P measurement libraries to write the files and by the SCOUT analyzer to read them. SIONlib [2] is employed to optimize parallel file I/O and avoid excessive numbers of files. The parallel replay analysis of the traces then follows, and an augmented analysis report is collated and written using the new version of CUBE.

Finally, the new Scalasca analysis report examiner, using the latest version of the CUBE utilities and GUI, is only superficially distinguished by the new report format (with a different file extension). Old and new analysis reports have very similar hierarchical structures of metrics, regions/call-paths and nodes/processes/threads, however, for reading and writing efficiency a binary file format is used for metric severity values rather than the previous XML. The new format also provides greater flexibility and functionality, e.g., for storing either inclusive or exclusive metric severity values. Notably, the Score-P measurement system does not yet provide any topological information for processes and threads nor compute nodes and processors, either in its own summary analysis reports or as part of the trace metadata definitions.

The first release of the new generation of Scalasca — version 2.0 — supports Score-P 1.2 and was co-released in August 2013. Measurements from this are refered to as Scalasca2 (S2) in comparison to the latest first generation Scalasca 1.4.3 release (S1) of March 2013.

3 Experimental Setup

3.1 Machines

It is of interest to compare these Scalasca generations on the two production HPC resources at Jülich Supercomputing Centre[1], since they offer quite different execution environments.

Juropa[2] is a high-performance cluster of 3288 compute nodes each with 24 GB of memory and dual quad-core Intel X5570 (Nehalem-EP) processors supporting 2-way hardware threading, connected via a non-blocking fat-tree Infiniband QDR network. The operating environment provides Intel compilers and ParaStation MPI running under SuSE Linux with Lustre filesystems.

Juqueen[3] is a highly scalable IBM BlueGene/Q system consisting of 28 racks each with 1024 PowerPC A2 processors with 16 GB of memory and 16 cores supporting 4-way hardware threading, and a custom 5D torus interconnect. Applications running under Linux microkernels on compute nodes use IBM XL compilers and proprietary MPI library with GPFS filesystems.

To effectively utilise the available hardware resources, a maximum of 16 OpenMP threads is appropriate for Juropa compute nodes, whereas 64 OpenMP threads is more suitable for Juqueen compute nodes.

[1] `http://www.fz-juelich.de/ias/jsc/EN/`
[2] `.../Expertise/Supercomputers/JUROPA/JUROPA_node.html`
[3] `.../Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html`

3.2 Applications

Many HPC applications now use combined MPI and OpenMP parallelization, therefore the multi-zone versions of the NAS Parallel Benchmarks (NPB3.3-MZ-MPI) [8] which run on a wide range of computer systems were chosen as the basis for the comparison. There are three separate pseudo-application benchmark codes — BT-MZ (block tri-diagonal solver), SP-MZ (scalar penta-diagonal solver) & LU-MZ (lower-upper Gauss-Seidel solver) — written in Fortran77 to solve various sizes of computational fluid dynamics problems (refered to as 'classes') at varying scales. Larger problem classes take longer to execute and require more memory.

Notably, while an arbitrary number of OpenMP threads can be used when executing, LU-MZ has a fixed number of even-sized zones and can therefore only be configured with a maximum of 16 MPI processes, whereas BT-MZ and SP-MZ can be configured with up to 16384 MPI processes (for the largest problem class 'F'). SP-MZ has equally-sized zones, but BT-MZ's zones have unequal sizes and correspondingly varying amounts of computation.

BT-MZ incorporates a static load balancing capability, varying the number of OpenMP threads for each process, however, this is ineffective when the threads are confined to the same processor on BlueGene systems (or same compute node on Juropa), and to avoid detrimental over-subscription such load balancing must be disabled. BT-MZ class F runs on Juqueen with the maximum 16384 MPI processes, though execution efficiency drops substantially with more than 4096 processes (each with 64 OpenMP threads). Analysis of the performance of BT-MZ execution with Scalasca has shown that at larger scale computational imbalance deteriorates particularly severely in the `z_solve` routine [9].

There is insufficient memory available on Juqueen compute nodes to run SP-MZ class F, however, SP-MZ class E runs and scales well with up to 4096 MPI processes (each with 64 OpenMP threads). With a maximum of only 16 MPI processes, LU-MZ is hardly suitable for systems like BG/Q where the smallest allocatable partition comprises 32 processors. Nevertheless, 16-process versions of all three benchmarks were configured for class D and run successfully on Juqueen as well as a Juropa partition of 16 compute nodes.

For reference we take the (wallclock) execution time reported for each benchmark kernel, which is relatively stable and not subject to variations in starting and initializing processes and their final completion. As well as uninstrumented optimized executables, additional executables for measurement were also prepared using the instrumenters from Scalasca 1 & 2. In each case, the instrumenter directed the respective compiler to insert instrumentation calls at every source routine entry and exit, as well as using OPARI2 for source-code instrumentation of OpenMP directives and linking the MPI+OpenMP measurement libraries. (In the course of subsequent measurement iterations, measurement control commands were incorporated in LU-MZ sources and measurement filter files generated for BT-MZ.)

To avoid performance variations that may arise from hardware partitions and sets of compute nodes, each experiment consisted of a series of measurements as well as reference executions in the same batch job using the same allocated compute resources. Parallel filesystems ($WORK) are required for efficient file I/O for tracing experiments, however, they are non-dedicated resources such that variability in performance is unavoidable.

4 Results

Apart from functionality known not to be available yet from Score-P, initial comparison of analysis reports produced by Scalasca 1 & 2 verified their correctness and completeness (with only minor issues). Separate series of experiments were undertaken to compare measurement and analysis overheads and scalability of the two versions.

4.1 Measurement Overhead

For each of the three benchmarks, Class D versions for 16 MPI processes were prepared on both Juropa and Juqueen, and these fully-optimised, uninstrumented executables run to have reference execution times, reported in Tab. 1.

Compilation and linking times of approximately 39–43 seconds on Juqueen increase to 45–47 seconds when instrumenting with both versions of Scalasca. On Juropa, compilation and linking times of 14–16 seconds increase to 37–43 seconds with Scalasca1, and 31–35 seconds with Scalasca2. Both versions of Scalasca use OPARI2 to process source files containing OpenMP, direct the compilers to insert routine entry and exit instrumentation calls, and additionally link the measurement libraries. Since shared libraries are available with Scalasca2 on Juropa, dynamic linking is faster than static linking done by Scalasca1. Although recompilation of the application is required, the additional time is moderate and the instrumented executables can be configured for different measurements.

An initial set of summary measurements were taken and scored by Scalasca to determine whether adjustments would be desirable or required, e.g., to specify buffer sizes for tracing measurements. Tab. 1 includes the maximum trace buffer capacity requirements (max_tbc) per-thread for Scalasca1 and per-process for Scalasca2. While there are no serious issues for SP-MZ, the score reports identify that traces produced by BT-MZ and LU-MZ would be too large to fit in available memory during measurement. The BT-MZ measurements also had kernel execution times much larger than the reference executions, by 1–2x for Scalasca1 and 2–4x for Scalasca2, indicating significant measurement dilation.

The BT-MZ score reports identified several user-level source routines that were executed very frequently and consequently would be responsible for the vast majority of the trace buffer capacity requirements (as well as ultimate trace size). These routines are short-running and tend to have disproportionately large measurement dilation, while not contributing to analysis of MPI or OpenMP parallel efficiency issues. Although the Intel compiler on Juropa appended an extra underscore character to the routine names, the exact same set of routines and scoring assessment applied for the IBM XL compiler on Juqueen. Measurement

Table 1. Reference execution times for NPB3.3-MZ-MPI benchmark kernels (class D, 16 processes) and characteristics of default and revised Scalasca 1&2 measurements on Juropa and Juqueen. Revised configurations using a measurement filter file for BT-MZ and explicitly instrumented measurement pause after 20 timesteps for LU-MZ.

	Juropa			Juqueen		
	16x16 BT-MZ	16x16 LU-MZ	16x16 SP-MZ	16x64 BT-MZ	16x64 LU-MZ	16x64 SP-MZ
Reference execution time [sec]	177.11	280.29	153.49	315.46	321.83	319.18
S1 default measurement time [sec]	378.20	258.05	151.39	907.63	333.24	408.32
S1 revised measurement time [sec]	184.41	272.79		375.72	333.29	
S2 default measurement time [sec]	581.10	265.08	153.57	1619.38	335.27	304.27
S2 revised measurement time [sec]	181.57	300.01		348.00	335.17	
S1 default measurement dilation [%]	114	7.9	1.4	188	3.5	27.9
S1 revised measurement dilation [%]	4.1	2.7		19.1	3.6	
S2 default measurement dilation [%]	228	5.4	0.1	413	4.2	4.7
S2 revised measurement dilation [%]	2.5	7.0		10.3	4.1	
S1 default thread max_tbc [MB]	13,488	6,396	74	6,438	1,603	119
S1 revised thread max_tbc [MB]	31	602		31	86	
S2 default process max_tbc [MB]	181,174	21,230	2,321	182,249	21,692	4,991
S2 revised process max_tbc [MB]	365	3,120		1,441	1,988	

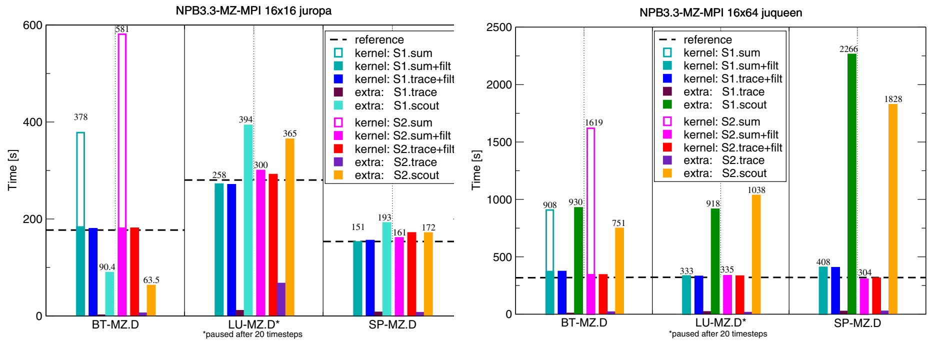


Fig. 1. NPB3.3-MZ-MPI benchmark class D experiments on Juropa (*left*) and Juqueen (*right*) comparing measurement dilation of execution kernels and additional time associated with trace collection and SCOUT trace analysis: for each benchmark, Scalasca1 left of dotted line and Scalasca2 on right. Black dashed lines show the uninstrumented reference kernel execution time, the first bar in each set is the kernel execution time for summary measurement (with and without filter for BT-MZ), with the following three bars separating kernel execution time for trace measurement, generation and analysis times.

filter files listing the names of these routines were therefore prepared and applied for subsequent summary and tracing measurements.

In comparison, while LU-MZ measurements showed acceptable dilation of up to 8% on Juropa and less than 5% on Juqueen, the LU-MZ score reports identified an OpenMP flush directive in the `sync_left` routine (`syncs.f:27`) that would be responsible for prohibitive trace buffer capacity requirements. Although it is possible to direct the Scalasca instrumenter not to instrument OpenMP flush directives, that would then compromise the subsequent analysis since OpenMP flush time would not be distinguished from computation time. With over 15% of total execution attributed to OpenMP flush time (and almost entirely from this particular one of the 5 flush directive use locations), it is desirable to retain this key part of the execution performance analysis. In this case, it is preferable to only measure a subset of the 300 LU-MZ.D timesteps, by explicitly inserting a conditional pause region in the source code of the timestep loop using the provided measurement control APIs. For LU-MZ.D, additional instrumented executables were prepared where measurement was paused after 20 timesteps, and corresponding trace buffer capacity requirements thereby reduced substantially.

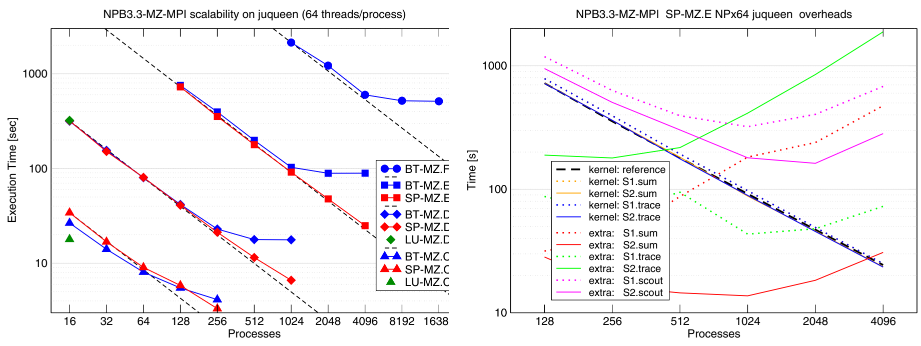
Measurement dilation and execution overheads for experiments with Scalasca 1 & 2 are shown in Fig. 1. Although unfiltered BT-MZ measurements suffer significant dilation, the filters employed to reduce traces to managable sizes are also effective in bringing dilations down to acceptable levels (below 20% on juqueen and only 7% on juropa). As well as dilated execution time of the measured application, trace generation and subsequent analysis with the SCOUT analyzer require compute nodes to be allocated for additional time. (Time for collating and writing summary measurements is negligible and not shown.) At this scale, trace generation times are small (even for the larger LU-MZ trace files), however, the trace analysis times are found to be highly dependent on the application and computer system (though Scalasca2 analysis is generally slightly faster than Scalasca1). If memory was available to trace the full 300 LU-MZ.D timesteps, rather than only 20, times for trace generation and analysis would be more than ten times longer.

4.2 Scalability

Execution times of the three benchmarks (classes C, D, E & F) with different numbers of MPI processes (one per processor) on Juqueen are shown in Fig. 2(a). In comparison to their execution times on 128 processors, with 4096 processors (i.e., 32 times as many) SP-MZ.E demonstrates an excellent speed-up of 29 whereas BT-MZ.E has a speed-up of only 8.4.

While the excellent speed-up characteristics of SP-MZ.E on Juqueen make its execution performance analysis less important for tuning, the absence of scaling artifacts also suggests it to be preferable when examining and comparing the scaling efficiencies of performance analysis tools.

Fig. 2(b) shows the SP-MZ.E kernel execution times for summary and tracing measurements with Scalasca1 and Scalasca2 (blue and orange lines,



(a) Kernel execution times (b) SP-MZ.E measurement experiments

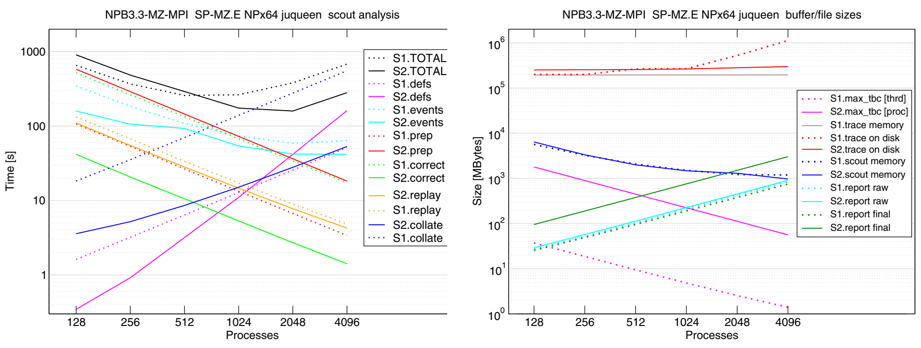
Fig. 2. Executions times on Juqueen BG/Q with 64 OpenMP threads per MPI process. (a) NPB3.3-MZ-MPI kernel execution times for the benchmark classes C (*triangle*), D (*diamond*), E (*rectangle*) & F (*circle*). (b) SP-MZ.E measurement dilation and execution overheads with Scalasca1 (*dotted lines*) & Scalasca2 (*solid lines*).

indistinguishable for different experiment types) and extra time associated with measurement and analysis. From comparison with the uninstrumented reference SP-MZ.E kernel execution times (black dashed line), measurement dilation was a maximum of 11% with Scalasca1 and 7% with Scalasca2 (compared with 28% and 5%, respectively, for SP-MZ.D_16x64 in Tab. 1). The larger dilation from Scalasca1 may be attributed to additional callpath consistency checking and metrics calculated.

Summary measurement finalization times (red lines in Fig. 2(b)) increase roughly linearly with the number of processes for which measurements are written to the analysis report. These are notably larger for Scalasca1 due to the extra metrics, along with the additional metadata describing the hardware and application topologies (not yet provided by Scalasca2), and report collation and writing time becomes significant at scale.

Trace generation times (green lines in Fig. 2(b)) similarly increase with the number of processes for which event data is written. Since Scalasca2 doesn't yet use the much more efficient SIONlib, creation and writing of trace files is currently considerably slower and less scalable than Scalasca1, and prohibitive for thousands of processes. Whereas Scalasca2 creates two files for every thread, Scalasca1 uses SIONlib to create only one file per I/O bridge on BG/Q and write thread data aligned to filesystem blocks as sections within these multifiles. As a result Scalasca2/OTF2 traces on disk only grow slowly from 246 GB for 128 processes to 292 GB with 4096 processes, whereas Scalasca1/SIONlib traces which are initially 25% smaller grow to be about four times larger than their counterparts (shown in red in Fig. 3(b)).

Trace analysis times (magenta in Fig. 2(b)) follow the same trend for both versions of Scalasca's automated trace analyzer SCOUT: good scalability for up to 512 processes (matching that of SP-MZ.E) is reversed for larger configurations. The time breakdown in Fig. 3(a) shows that for Scalasca1 this is primarily



(a) Trace analysis time breakdown (b) File sizes and memory requirement

Fig. 3. Scaling of SP-MZ.E tracing experiments on Juqueen with Scalasca1 (*dotted lines*) & Scalasca2 (*solid lines*). (a) Breakdown of time in stages of SCOUT automated trace analysis. (b) Trace buffer content and file size on disk, trace analyzer memory requirement, and analysis report size.

explained by the time to collate and write the linearly growing analysis report sizes, whereas for Scalasca2 it is reading the trace metadata definitions that dominates at scale. At smaller scales, Scalasca2 is dominated by time to preprocess the already loaded trace data. Scalasca2 trace replay analysis is found to be consistently faster than its predecessor.

While Fig. 3(b) shows that the size of the largest trace buffers decreases linearly, the total amount of trace event data in memory buffers is largely independent of the number of processes (approx. 190 GB) and disk space for the resulting trace files is somewhat larger. Also both versions of the Scalasca trace analyzer require similar amounts of memory and produce raw trace analysis reports of comparable size.

To be efficient and flexible, analysis reports written directly by the Scalasca measurement libraries and parallel trace analyzers contain a minimal set of unstructured metrics. These intermediate Scalasca2 reports in CUBE4 format are 4–8% larger than Scalasca1 reports in CUBE3 format. Additional metrics are derived and a hierarchy of metrics created during subsequent offline post-processing. The final analysis reports are four times larger with Scalasca2, but processed 4–5 times faster by the scoring utility, since they store inclusive metric values for more efficient reading rather than the exclusive values supported by Scalasca1. Analysis report sizes and post-processing times are proportional to the complexity of the raw reports, determined by the number and type of call-paths but also by the number of processes (and threads per process). Whereas the Scalasca1 metric remapper utility uses hard-coded metric derivations and mappings, the Scalasca2 version is configurable via specification files. Remapping the SP-MZ.E trace analysis reports for 4096 processes takes just over 10 minutes with Scalasca1, but roughly 20 times longer with Scalasca2.

5 Conclusion

Basic functionality of the new version (2.0) of the Scalasca toolset has been validated with the three MPI+OpenMP codes of the NPB3.3-MZ-MPI benchmark suite on IBM BlueGene/Q and a large Linux cluster. Its performance and scalability, particularly measurement dilation and overheads associated with trace generation and analysis, have been examined and found to be comparable (and in many cases better) than its predecessor. Score-P/OTF2 tracefile handling needs to be improved and future versions are expected to use SIONlib to improve scalability. As well as supporting additional HPC platforms, missing functionality such as topology descriptions are also expected to be incorporated. However, currently the most critical limitation of Scalasca2 is the very slow and inefficient analysis report post-processing (remapper), which is inconvenient both for experiments with complex call-trees or large numbers of processes/threads.

References

1. Chung, I., Walkup, R., Wen, H., Yu, H.: MPI performance analysis tools on BlueGene/L. In: Proc. ACM/IEEE SC 2006 Conference on High Performance Networking and Computing, Tampa, FL, USA. ACM Press (November 2006)
2. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel I/O to task-local files. In: Proc. ACM/IEEE SC 2009 Conference Portland, OR, USA (November 2009), `http://www.fz-juelich.de/jsc/sionlib/`
3. Geimer, M., Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Mohr, B.: The *Scalasca* performance toolset architecture. Concurrency and Computation: Practice and Experience 22(6), 702–719 (2010), `http://www.scalasca.org/`
4. GWT-TUD GmbH: VAMPIR. Technische Universität Dresden, Dresden, Germany, `http://www.vampir.eu/`
5. Iwainsky, C.: an Mey, D.: Comparing the usability of performance analysis tools. In: César, E., Alexander, M., Streit, A., Träff, J.L., Cérin, C., Knüpfer, A., Kranzlmüller, D., Jha, S. (eds.) Euro-Par 2008 Workshops. LNCS, vol. 5415, pp. 315–325. Springer, Heidelberg (2009)
6. Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A.D., Nagel, W.E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S.S., Tschüter, R., Wagner, M., Wesarg, B., Wolf, F.: Score-P – A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Proc. 5th Parallel Tools Workshop, Dresden, Germany, pp. 79–91. Springer (2012), `http://dx.doi.org/10.1007/978-3-642-31476-6_7`, `http://www.score-p.org/`
7. Mohr, B., Wylie, B.J.N., Wolf, F.: Performance measurement and analysis tools for extremely scalable systems. Concurrency and Computation: Practice and Experience 22(16), 2212–2229 (2010)
8. Van der Wijngaart, R.F., Jin, H.: NAS Parallel Benchmarks, Multi-Zone versions. Tech. Rep. NAS-03-010, NASA Ames Research Center, Moffett Field, CA, USA (July 2003), `http://www.nas.nasa.gov/Software/NPB/`
9. Wylie, B.J.N.: Parallel performance measurement & analysis scaling lessons. In: Proc. SC 2012 Workshop on Extreme-Scale Performance Tools, Salt Lake City, UT, USA (November 2012), `http://juser.fz-juelich.de/record/128166`