Integrated runtime measurement summarisation and selective event tracing for scalable parallel execution performance diagnosis

Brian J. N. Wylie[1], Felix Wolf[1,2], Bernd Mohr[1], and Markus Geimer[1]

[1] John von Neumann Institute for Computing (NIC), Forschungszentrum Jülich, 52425 Jülich, Germany
[2] Dept. Computer Science, RWTH Aachen University, 52056 Aachen, Germany
{b.wylie, f.wolf, b.mohr, m.geimer}@fz-juelich.de

Abstract. Straightforward trace collection and processing becomes increasingly challenging and ultimately impractical for more complex, long-running, highly-parallel applications. Accordingly, the SCALASCA project is extending the KOJAK measurement system for MPI, OpenMP and partitioned global address space (PGAS) parallel applications to incorporate runtime management and summarisation capabilities. This offers a more scalable and effective profile of parallel execution performance, for an initial overview and to direct instrumentation and event tracing to the key functions and callpaths for comprehensive analysis. The design and re-structuring of the revised measurement system are described, highlighting the synergies possible from integrated runtime callpath summarisation and event tracing for scalable parallel execution performance diagnosis. Early results from measurements of 16,384 MPI processes on IBM BlueGene/L already demonstrate considerably improved scalability.

1 KOJAK/SCALASCA event tracing and analysis

The KOJAK toolset provides portable automated measurement and analysis of HPC applications which use explicit message-passing and/or implicit shared-memory parallelisations with MPI, OpenMP and PGAS [2, 3]. Via interposition on library routines, preprocessing of source code directives/pragmas, or interfacing with compilers' function instrumentation, a comprehensive set of communication and synchronisation events pertinent to the execution of a parallel application can be acquired, augmented with timestamps and additional metric measurements, and logged in trace files. These time-ordered event traces from each application thread are subsequently merged into a global time-ordered trace for analysis, via automatic rating of performance property patterns or interactive time-line visualisation.

Despite the demonstrated value of the event tracing approach, using KOJAK, VAMPIR [4], DiP/Paraver [5] or commercial alternatives such as Intel Trace Collector and Analyzer, a key limitation is the trace volume which is directly proportional to granularity of instrumentation, duration of collection, and number of threads [6]. Multi-process and multi-thread profiling tools such as MPIP [13], OMPP [14] and proprietary equivalents, avoid this limitation by aggregating execution statistics during measurement and producing performance summaries at completion. A variety of tracing and profiling options have been incorporated by the TAU toolkit [7], including flat (function), specifiable-depth callpath and calldepth profiling. Measurement tools can generally exploit library interposition, via re-linking or dynamic library loading, to track MPI usage, however, implicit parallelisation with OpenMP and function tracking typically require dynamic instrumentation [8] or re-compilation to insert special instrumentation.

SCALASCA is a new project which is extending KOJAK to support scalable performance measurement and analysis of large-scale parallel applications, such as those consisting of thousands of processes and threads. Parallelisation of post-mortem trace analysis via replay of events and message transfers required reconstruction of the trace measurement and automated analysis foundation [9]. This is being complemented with runtime summarisation of event measurements into low-overhead execution callpath profiles, which will also be used to reconfigure instrumentation and measurements or direct selective event tracing. Reports from both runtime summarisation and trace analysis will use the same CUBE format, so they can also be readily combined, for presentation and investigation with the associated browser [10]. Ultimately, improved integration of instrumentation, measurement and analyses will allow each to be progressively refined for large-scale, long-running, complex application executions, with automation providing ease of use.

After describing the synergies possible from integrating runtime callpath profiling and event tracing, the design of the revised measurement system is introduced, and specific usability and scalability improvements that have been incorporated are detailed, followed by discussion of initial results with the prototype revised implementation of event tracing which demonstrate its effectiveness at a range of scales.

2 Runtime measurement summarisation

An approach without the scalability limitations of complete event tracing is runtime measurement summarisation. As each generated event is measured, it can be immediately analysed and incorporated in a summary for events of that type occurring on that program callpath (for that thread). Summary information is much more compact than event traces, with size independent of the total collection duration (or the frequency of occurrence of any function): it is equivalent to a local profile calculated from the complete event trace, and combining summaries produces a global callpath profile of the parallel execution.

For measurements which are essentially independent for each process, such as interval event counts from processor hardware counters, runtime summarisation can effectively capture the profile without the overhead of rendering a bulky vector of measurements. On the other hand, performance properties related to inter-process interaction, such as the time between when a message was sent and available to the receiver and its eventual receipt (i.e., “late receiver”), can only be determined in a portable way by combining disjoint measurements that is not practical at runtime. Fortunately, the inter-process performance properties are generally specialised refinements of the process-local ones available from runtime summarisation.

Doing the local analysis at runtime during measurement, and in parallel, reduces the need for large files and time-consuming post-processing and results in a timely initial overview of the execution performance.

Runtime measurement processing and summarisation also offers opportunities to decide how to deal with each event most effectively as it is generated. Frequently encountered events may be candidates to be simply ignored, due to the overhead of processing measurements for them. Other events may have a very variable cost which is typically small enough to be negligible but occasionally significant enough to warrant an explicit detailed record of their occurrence.

Alternatively, a profile summary from which it is possible to determine how frequently each function is executed, and thereby assess their importance with respect to the cost of measurement, can be used as a basis for selective instrumentation which avoids disruptive functions. Subsequent measurements can then benefit from reduced perturbation for more accurate profiling or become suitable for complete event tracing.

Runtime measurement summarisation therefore complements event tracing, providing an overview of parallel execution performance which can direct instrumentation, measurement and analysis for more comprehensive investigation.

3 Integration of summarisation and tracing

An integrated infrastructure for event measurement summarisation and tracing offers maximum convenience, flexibility and efficiency. Applications instrumented and linked with the measurement runtime library can be configured to summarise or trace events when measurement commences, and subsequent measurements made without rebuilding the application. It also becomes possible to simultaneously combine both approaches, with a general overview profile summary refined with analysis of selective event traces where they offer particular insight.

Along with the practical benefit of maintaining a single platform-specific measurement acquisition infrastructure, sharing measurements of times, hardware counters and other metrics avoids duplicating overheads and potential access/control conflicts. It also facilitates exact comparison of aggregate values obtained from both approaches.

Some form of runtime summarisation is probably always valuable, perhaps as a preview or compact overview. Metrics calculated from hardware counter measurements are generally most effectively captured in such summaries. Extended summaries with additional statistics calculated may be an option. Only in the rare case where the runtime overhead should be reduced to an absolute minimum is it expected that summarisation might be completely disabled.

Unless it can be readily ascertained that the application's execution characteristics are suitable for some form of event tracing, the default should be for tracing to initially be inactive. When activated, simply logging all events would provide the most complete execution trace where this was desired (and from which a summary profile could be calculated during postprocessing analysis). Alternatively, selective tracing may be based on event characteristics (such as the event type or a measurement duration longer than a specified threshold), or based on analysis from a prior execution (e.g., to filter uninteresting or overly voluminous and obtrusive measurements).

Furthermore, the availability of measurements for the entry of each new function/region frame on the current callstack, allows for late determination of whether to include them in an event trace. For example, it may be valuable to have a trace of all communication and synchronisation events, with only function/region entry and exits relevant to their callpaths (and all others discarded at runtime). The callstack and its entry measurements can be tracked without being logged until an event of interest is identified (e.g., by its type or duration), at which point the (as yet unlogged) frame entry measurements from the current callstack can be used to retroactively log its context (and mark the associated frames such that their exits will also be logged), while new frames subsequently encountered remain unlogged (unless later similarly identified for logging).

To have the most compact event traces, only message transfers need to be logged with their callpath identifier. Non-local performance properties subsequently calculated from post-mortem analysis of the traced message transfer events can then be associated with the local performance properties for the same callpaths already generated by runtime summarisation.

If desired, separate dedicated libraries for summarisation and tracing could also be provided and selected during application instrumentation preparation.

4 Implementation of revised measurement system

KOJAK's measurement runtime system formerly was based on an integrated event interfacing, processing and logging library known as EPILOG [11]. Files containing definitions and event records were produced in EPILOG format, and manipulated with associated utilities. Execution traces can have performance properties automatically analysed, or can be converted to other trace formats for visualisation with third-party tools.

The EPILOG name, file format and utilities are retained in the revised design of the measurement system, but only for the logging/tracing-specific component of a larger integrated summarisation and tracing library, EPIK, as shown in Figure 1. Event adapters for user-specified annotations, compiler-generated function instrumentation, OpenMP instrumentation, and the MPI library instrumentation are now generic, rather than EPILOG/tracing specific. Similarly, platform-specific timers and metric acquisition, along with runtime configuration and experiment archive management, are common EPIK utility modules. A new component, EPITOME, is dedicated to producing and manipulating totalised measurement

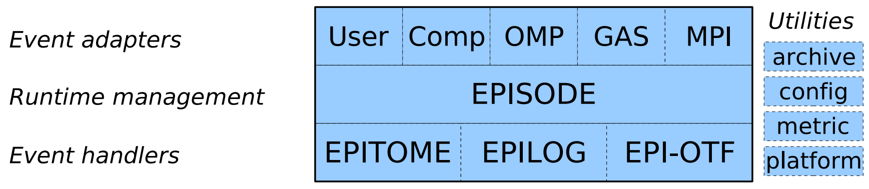

Fig. 1. EPIK runtime library architecture.

summaries. Both EPILOG and EPITOME share a common runtime management system, EPISODE, which manages measurement acquisition for processes and threads, attributes them to events, and determines which summarisation and/or logging subsystems they should be directed to (based on the runtime measurement configuration). Additional auxilliary event processing and output can also be incorporated as EPIK back-end event handler modules.

In addition to restructuring the measurement system, various usability and scalability improvements have been incorporated, to be able to manage measurements collected from thousands of processes.

4.1 Usability improvements

To facilitate diverse measurement collections and analyses, and avoid the clutter of multiple (and perhaps thousands of) files appearing in the program's working directory, a new experiment archive directory structure has been introduced. A unique directory is created for each measurement execution to store all of the raw and processed files associated with that measurement and its analysis. Instead of applying each KOJAK tool to files of the appropriate type, the tools can now also accept the name of the experiment archive directory, from which the appropriate files are transparently located and into which new files are deposited. This new structure makes it easier for the tools to robustly determine and maintain the integrity of the experiment measurement/analyses, and should also be easier for users to manage (e.g., when they wish to move an experiment to a different system for storage or analysis).

On-the-fly file compression and decompression [12] reduces the size of experiment archives, with an additional bonus in the form of reduced file reading and writing times (despite additional processing overheads).

In addition to a library which can be used by external tools to read and write EPILOG traces, utilities are provided to convert traces to and from the formats used by other tools, such as VAMPIR and Paraver. Furthermore, as an alternative to post-mortem trace conversion, experimental support has been incorporated within EPIK to directly generate traces in OTF format [15].

4.2 Scalability improvements

EPILOG traces were previously written from each thread's collection buffer into temporary files, which were merged according to event timestamp order into process rank traces and finally a global trace file at measurement finalisation. Re-reading and re-writing these large trace files was required to produce a sequential trace that the sequential analysis tool could handle. Furthermore, additional scans through each trace were required to locate the definition records interspersed within the files so that they could be globalised and written as a header in the merged file. Although this merging was often initiated automatically, and the thread stage was partially parallelised, it was a notable bottleneck in the combined measurement and analysis process which was extremely sensitive to filesystem performance. Fortunately, parallel trace analysis has no need for merged trace files, since the analysis processes read only the trace files that they require [9].

The traces written by each thread can therefore be written directly into the experiment archive, from where the subsequent analysis processes can access them. These event traces are written using process-local identifiers for regions, communicators, threads, etc., which will later need to be converted to a globally consistent set of identifiers for a unified analysis. Definitions which were previously interspersed with event records in the traces are now handled separately.

As an interim solution, these local definitions have been written to files to be unified into a global definitions file and associated local–global mapping files via postprocessing. These files are much smaller than the corresponding event traces, and can be unified quite efficiently by the separate unifier, however, creating large numbers of (small) intermediate files has been found to be inefficient.

Generation of the global definitions and associated local–global identifier mappings is required for unified analysis, and although it is a predominantly sequential operation, it is advantageous for it to be done on conclusion of the parallel measurement. Instead of each process rank writing local definitions to file(s), the buffers can be sent to rank 0 to produce a global set and associated identifier mappings. Post-mortem trace analysis requires this information to be filed along with the traces in the experiment archive, however, the runtime measurement summarisation can immediately exploit the returned identifier mappings to directly produce a unified measurement summary.

Although all of the definitions are required for the complete analysis report, identifiers do not require to be globalised when they are common to all processes or can be implied, such as the measurement metrics (time and hardware counters) and machine/node & process/thread identifiers. The remaining analysis is for callpaths, consisting of lists of region identifiers; unified analysis requires globalisation of these callpath identifiers.

Callpaths can be specified as node-region-id and parent-callpath-id records, so that only tail-segments need to be defined and callpaths are reconstructed by combining segments from each tail node via its parents' nodes to the root (which has a null parent-callpath-id). These can be added to the existing local definitions in the buffers to be unified by rank 0, or specified and unified separately.

Tracking the current callpath is part of the EPISODE functionality, whereas EPITOME maintains the set of local callpaths according to which measurements are summarised, therefore it is straightforward to provide a complete set of (local) callpaths on measurement conclusion. Provision of the global callpath set and local–global callpath identifier mappings also allows these to be used in the post-mortem trace analysis, avoiding a scan through the trace to determine the local callpaths and their subsequent unification. Even in the case where EPITOME measurement summarisation is not performed, there are still advantages from maintaining callpaths and associated visit counts.

The primary use is likely to be the use of callpath visit counts to threshold the number of times paths (events) are traced. In conjunction with the callstack of (region-entry) measurements, region enter/exit events can be tracked until an event of interest (such as a message transfer) indicates that the current buffer of events should be logged, such that uninteresting callpaths are effectively pruned from the trace. Alternatively, message transfers may be tagged with (local) callpath identifiers, allowing them to be efficiently traced in isolation from the region enter/exit events that otherwise determine callpath context. These approaches reduce the overhead of tracing frequently-executed, intrusive and/or uninteresting callpaths, making tracing and subsequent trace analysis more effective.

5 Results

To evaluate the effectiveness of some of the changes to the KOJAK measurement system, a number of tracing experiments have been performed at a range of scales with the current prototype implementation and the prior version.

Measurements were taken on the Jülicher BlueGene/L system (JUBL), which consists of 8,192 dual-core 700 MHz PowerPC 440 compute nodes (each with 512 MBytes of memory), 288 I/O nodes, and IBM p720 service and login nodes each with eight 1.6 GHz Power5 processors [16]. The parallel measurements were made on a dedicated compute partition, whereas the sequential post-processing steps ran on the lightly loaded login node.

ASC benchmark SMG2000 [17] is a parallel semi-coarsening multigrid solver, and the MPI version performs many non-nearest-neighbor point-to-point communication operations (and only a negligible number of collective communication operations). In an investigation of *weak scaling* behaviour, a fixed $64 \times 64 \times 32$ problem size per process with five solver iterations was configured, resulting in a nearly constant application run-time as additional CPUs were used: uninstrumented execution times are shown with open diamonds in Figure 2, along with a breakout of the wall time spent in the parallel solver as open triangles.

Instrumented versions of SMG2000 were prepared with the prior and current development versions of the KOJAK measurement system, and the times for running these are also shown in Figure 2: the lighter solid diamonds are the older version and the darker solid diamonds the latest version, which is more than an order of magnitude faster at the larger scales.

In each case, measurement was done using 100MByte trace buffers on each process to avoid intermediate buffer flushes to file which would otherwise seriously impact performance during measurement. The time taken by the parallel solver is the same for both versions when instrumented and measured (shown with solid right triangles) and found to be dilated less than 15% compared to the uninstrumented version (open left triangles), which is generally considered acceptable. Similarly, the number of events traced for each process is also identical in both versions, and slowly increases with the total number of processes: the crosses in Figure 2 are the mean number of events per process (in thousands), with the vertical extents corresponding to the range (which grows to more than ±50% of the mean). The aggregate number of events traced therefore increases somewhat faster than linearly, to over 40,000 million events in the 16,384-process configuration, and this manifests in the total sizes of the measurements when archived on disk. In its new compressed form, the corresponding experiment archive totals almost 230 GBytes, whereas the former uncompressed trace files are around 2.5 times larger. On-the-fly compression is a factor in the improved performance, however, the most significant gain is from avoiding trace re-writing and merging (which also makes trace writing times rather more deterministic).

The benefits of the new approach are especially evident when post-processing the traces. The prior version of the KOJAK measurement system performs a semi-parallel thread trace merging when experiments are archived, which is followed

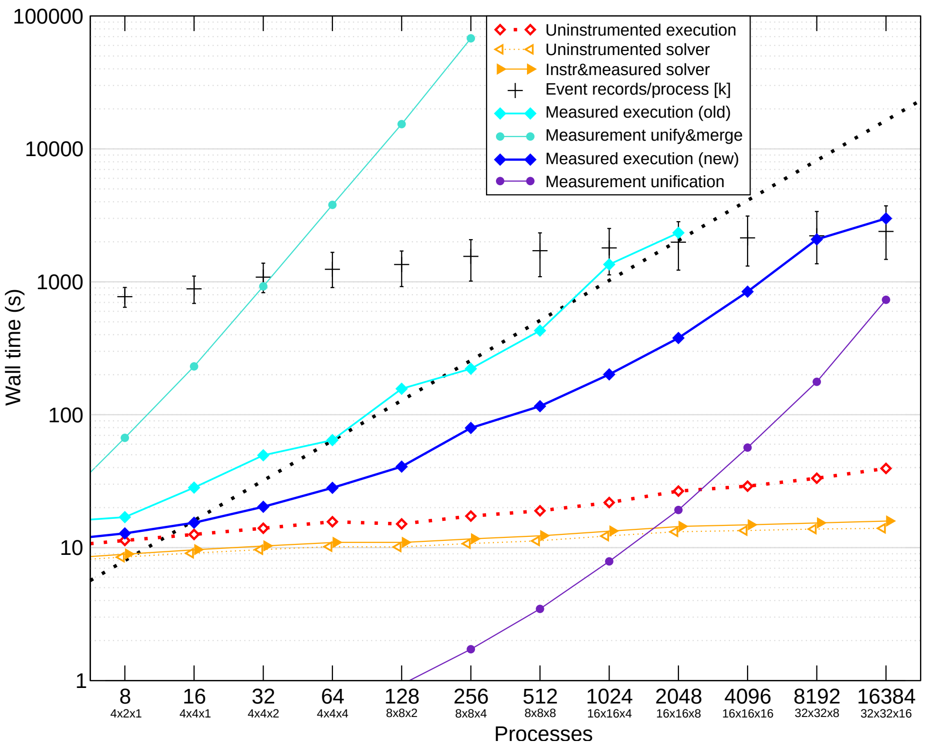

Fig. 2. SMG2000 trace collection and post-processing times with prior and current development versions of KOJAK at a range of scales on Blue Gene/L, compared with the uninstrumented execution. (Log-log scale with dotted line marking linear scaling.)

by a separate sequential process trace unification and merge: the light circles in Figure 2 show how this quickly becomes impractical for larger scales. By writing local definitions and thread event traces directly into the experiment archive, followed by a separate sequential unification of just the definition records and creation of local–global identifier mappings (the darker circles), the new version scales much more favourably.

These improvements combined demonstrate trace measurement now scaling to 16,384 processes, considerably beyond the prior practical limits. (The correspondingly more scalable analysis of the new traces is further motivation [18].)

6 Further and future work

Definition unification has subsequently been incorporated within the measurement finalisation, where gathering and unifying the local definitions both avoids the need to create definitions files for each process and the separate (sequential) unification step, thereby further improving measurement performance. Only when traces are produced for post-mortem analysis, is it necessary for the unified global definitions and identifier mappings to be written for each rank.

Runtime callpath summary data similarly needs to be gathered by the master process, so that it can be written as a unified report. The local callpath identifiers used by each process to store its measurements must therefore be mapped to those of the global call-tree, using the mappings returned to each process rank by the master process for this purpose. Finally, serial report writing functionality has already been refactored from the CUBE library, allowing callpath measurement data to be streamed to file as it is gathered, measurement by measurement, without it needing to be previously stored in memory in its entirety.

The runtime analysis summary reports are being formatted for presentation and investigation with the same CUBE analysis browser used for the reports produced by the former sequential and new parallel automatic event trace analysers. Direct comparison will thereby be possible using the CUBE algebra utilities [10], and will facilitate determination of instrumented functions which are problematic, due to their frequency of execution or measurement overheads. Selective instrumentation or measurement configuration can then be employed to circumvent those functions (or callpaths) in subsequent performance measurement executions, to obtain the highest quality analysis in a reliable, scalable manner.

The effectiveness of the new measurement and analysis capabilities are being evaluated on a range of HPC systems and applications, particularly at large scale, where they are also being compared with other tracing and profiling tools. Even simple operations, such as creating a separate file for each process in the same directory, can become prohibitively expensive at the largest scales, suggesting a need for exploiting the system-specific hierarchy in the structure of the measurement archive. Similarly, coordination of file writing may benefit from being adapted to the capabilities of the underlying I/O and file system.

References

1. Forschungszentrum Jülich GmbH: SCALASCA: Scalable performance Analysis of Large-Scale parallel Applications. `//www.scalasca.org/`
2. Forschungszentrum Jülich GmbH (ZAM) and the University of Tennessee (ICL): KOJAK: Kit for Objective Judgement and Knowledge-based detection of performance bottlenecks. `//www.fz-juelich.de/zam/kojak/`
3. Wolf, F., Mohr, B.: Automatic Performance Analysis of Hybrid MPI/OpenMP Applications. J. Systems Architecture, 49(10–11). Elsevier (2003) 421–439
4. Nagel, W., Arnold, A., Weber, M., Hoppe, H.-C., Solchenbach, K.: VAMPIR: Visualization and Analysis of MPI Resources. Supercomputer, 63(1). (1996) 69–80
5. Labarta, J., Girona, S., Pillet, V., Cortes, T., Gregoris, L.: DiP: A Parallel Program Development Environment. Proc. 2nd Int'l EuroPar Conf. on Parallel Processing (Lyon, France), Lecture Notes in Computer Science 1124, Springer (1996) 665-674
6. Wolf, F., Freitag, F., Mohr, B., Moore, S., Wylie, B. J. N.: Large Event Traces in Parallel Performance Analysis. Proc. 19th Int'l Conf. on Architecture of Computing Systems (Frankfurt am Main, Germany) Lecture Notes in Informatics, P-81. Gesellschaft für Informatik (2006) 264–273
7. Shende, S. S., Malony, A. D.: The TAU Parallel Performance System. Int'l J. High Performance Computing Applications, 20(2). SAGE Publications (2006) 287–331
8. Cain, H. W., Miller, B. P., Wylie, B. J. N.: A Callgraph-based Search Strategy for Automated Performance Diagnosis. Concurrency and Computation: Practice and Experience, 14(3). (2002) 203–217
9. Geimer, M., Wolf, F., Knüpfer, A., Mohr, B., Wylie, B. J. N.: A Platform for Scalable Parallel Trace Analysis. Proc. 8th Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'06, Umeå, Sweden), Lecture Notes in Computer Science, Springer.
10. Song, F., Wolf, F., Bhatia, N., Dongarra, J., Moore, S.: An Algebra for Cross-Experiment Performance Analysis. Proc. 33rd Int'l Conf. on Parallel Processing (ICPP'04, Montreal, Quebec, Canada). IEEE Computer Society (2004) 63–72
11. Wolf, F., Mohr, B., Bhatia, N., Hermanns, M.-A.: EPILOG binary trace-data format, version 1.3 (2005) `//www.fz-juelich.de/zam/kojak/doc/epilog.pdf`
12. Gailly, J., Adler, M.: zlib general-purpose compression library, version 1.2.3 (2005) `//www.zlib.net/`
13. Vetter, J., Chambreau, C.: MPIP — lightweight, scalable MPI profiling (2005) `//www.llnl.gov/CASC/mpip/`
14. Fürlinger, K., Gerndt, M.: OMPP — A Profiling Tool for OpenMP. Proc. 1st Int'l Work. on OpenMP (IWOMP, Eugene, OR, USA) (2005)
15. Knüpfer, A., Brendel, R., Brunst, H., Mix, H., Nagel, W. E.: Introducing the Open Trace Format (OTF). Proc. 6th ICCS (Reading, UK), Part II, Lecture Notes in Computer Science 3992, Springer (2006) 526–533
16. The BlueGene/L Team at IBM and LLNL: An overview of the BlueGene/L supercomputer. Proc. SC2002 (Baltimore, MD, USA). IEEE Computer Society (2002)
17. Advanced Simulation and Computing Program: The ASC SMG2000 benchmark code. (2001) `//www.llnl.gov/asc/purple/benchmarks/limited/smg/`
18. Geimer, M., Wolf, F., Wylie, B. J. N., Mohr, B.: Scalable Parallel Trace-based Performance Analysis. Proc. 13th European PVM/MPI User's Group Meeting (Bonn, Germany), Lecture Notes in Computer Science 4192, Springer (2006) 303–312