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

Brian J. N. Wylie, Felix Wolf, Bernd Mohr, and Markus Geimer

John von Neumann Institute for Computing (NIC),
Forschungszentrum Jülich, D-52425 Jülich, Germany
kojak@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 KOJAK measurement system for MPI, OpenMP and SHMEM parallel applications is incorporating 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 comprehensive scalable parallel execution performance diagnosis.

1 KOJAK event tracing and analysis

The KOJAK toolkit provides portable automated measurement and analysis of HPC applications which use explicit message-passing and/or implicit shared-memory parallelisations with MPI, OpenMP and SHMEM [1, 2]. 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 generated, augmented with timestamps and additional metric measurements, and logged in trace files. These time-ordered event traces from each application thread and process are subsequently merged into a global time-ordered trace for analysis, either as an automatic rating of performance property patterns or interactive time-line visualisation. Despite the demonstrated benefits of the event tracing approach, a key limitation is the trace volume which is directly proportional to granularity of instrumentation, duration of collection, and number of threads/processes [3].

2 Runtime measurement summarisation

An approach without the scalability limitations of complete event tracing is runtime measurement summarisation, which has been extensively investigated by

the TAU project [4]. 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/process). 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 from combining 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 this straightforward 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 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 would 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 corresponding 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 (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 (until and unless similarly identified for logging).

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. Files containing definitions and event records were produced in EPILOG format, and manipulated with associated utilities (such as `elg_merge`). Execution traces can have performance properties automatically analysed with EXPERT, or 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. A new component,

EPITOME, is dedicated to producing and manipulating totalised measurement statistical summaries. Both EPILOG and EPITOME share a restructured common event processing and interfacing runtime system, EPISODE, which manages measurements for processes and threads, attributes them to generated events, and determines which summarisation and/or logging subsystems they should be directed to (based on the runtime measurement configuration).

To facilitate diverse measurement collections and analyses, and avoid the clutter of multiple 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 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).

5 Further and future work

Callpath summary profiles will be formatted for presentation and investigation with the same CUBE analysis browser used for the reports produced by the EXPERT automatic event trace analyser. Direct comparison will thereby be possible using the CUBE algebra utilities, 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 KOJAK capabilities are being evaluated on a range of HPC systems and applications, and will be compared with other profiling and tracing toolsets, such as TAU.

References

1. 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/`
2. Wolf, F., Mohr, B.: Automatic Performance Analysis of Hybrid MPI/OpenMP Applications. J. Systems Architecture, 49(10–11). Elsevier (2003) 421–439
3. Wolf, F., Freitag, F., Mohr, B., Moore, S., Wylie, B. J. N.: Large Event Traces in Parallel Performance Analysis. Proc. 8th Workshop on Parallel Systems and Algorithms (PASA'06, Frankfurt/Main, Germany). Lecture Notes of Informatics, Gesellschaft für Informatik. (to appear)
4. Schende, S. S., Malony, A. D.: The TAU Parallel Performance System. Int'l J. High Performance Computing Applications. SAGE Publications (to appear) `//www.cs.uoregon.edu/research/paracomp/papers/ijhpca05.tau/TAU_ACTS_SS.pdf`

A Experiment archive issues

The proposed new experiment archive directory contains subdirectories for (unmerged) definitions and event traces for each process (or alternatively for threads). These files may be analysed in parallel, or if they are merged into global files, then the unmerged files and their containing directory can be removed. Experiment archive directories are recognisable via a specific prefix in their names, with the remainder specifiable by the user when measurement commences. After the initial measurement collection in files on local/distributed filesystems, the experiment directory provides an archive for subsequent processing and analyses. The directory and its contents can be moved to other systems for storage or analysis, and removed when no longer required, using regular Unix commands.

During migration from the former trace-specific, unified measurement files for sequential analysis to the new multi-format, split files for distributed analysis, several transitional approaches are possible.

Each tool that formerly required a specific file of the corresponding type can be augmented to 'automatically' locate such files inside a specified experiment directory, and it should be expected that any files that are produced would also be placed in the experiment directory by default. If this is applied to the trace conversion tools (elg2vtf3 and elg2otf), that would require that the precise location of the resulting files be documented and that the path to that location be provided to readers such as VAMPIR (who can't be expected to understand the experiment archive directory layout).

An alternative would be to provide an option to elg_merge (via an environment variable) which reverts to the former behaviour: i.e., although the experiment archive directory is used internally, the resulting merged trace file is named as before directly in the (specified) global directory, after which the archive directory and its contents are removed.

To avoid the need to merge event traces as required by the current EPILOG reader/writer interface, it could be extended to also handle experiment directories and the split files contained therein, providing a unified view to tools using it, such as the format converters. (Note that when an EPILOG trace file is explicitly specified it should be considered a complete merged trace, whereas a trace file automatically located inside an experiment directory might not contain definitions if these have been split into a separate global definitions file.)

Alternatively, the old tools could be deprecated (with their use restricted to the former files and formats), and a different set of equivalent tools/utilities could be provided exclusively to handle the new experiment archive directory (and ignore older non-archive files).

The EARL/EXPERT analyser is expected to remain sequential, in due course being complemented with (and perhaps ultimately replaced by) a parallel analyser (PEARL/SEXPERT?) which would directly exploit unmerged trace files.

While the contents of the experiment archive directory are regular visible files (and subdirectories), it is instructive to consider the archive as an opaque object (which might be a database, for example). Direct modification or re-arrangement of its contents should therefore be discouraged, with access via a higher-level interface that doesn't expose such implementation detail.

Experience with the new experiment archive, particularly on systems with highly-parallel microkernel-based architectures, is required to evaluate its effectiveness and possible portability issues. Further, it should be included in an upcoming beta release

to provide an opportunity for others to use and familiarise with it (particularly in the context of third-party tools and utilities) and provide feedback.

Currently, the naming of the experiment archive directory (and its contents) are provisional, and existing EPILOG environment variables have been (ab)used to control renaming and relocation. Once internal naming issues have been resolved, the KOJAK USAGE documentation should be updated to reflect the new behaviour, to the extent that it affects what arguments are given to the various tools.

Some specific early questions:

Is an elg_merge option necessary to revert to the former naming behaviour? No, we only want to document/support the new behaviour, and solicit feedback on that. When desired, it's straightforward for users to copy/rename the new files to their old locations/names, i.e., `$ELG_PFORM_GDIR/$ELG_FILE_PREFIX.elg`.

Is the extended EPILOG reader/writer interface for the new archive directories expected to be appropriate for external tools? Yes, it should be, and if not we need to determine (and fix) this sooner rather than later.

Will external tools be able to locate the files they require from within the experiment archive directory? We need to adequately document the necessary parts of the new archive directory, in particular, the names of the files that are intended to be externally visible/usable, e.g., epik.elg, epik.cube, epik.vpt (or epik.vtf) and epik.otf. The OTF auxilliary/rank files created by EPIK (either produced directly by libepik or converted by elg2otf) should be in a dedicated EPIK/OTF subdirectory: the epik.otf index file may also be in that subdirectory, or directly in the EPIK archive directory itself.

Should external tools (such as VAMPIR) be encouraged to write their own files into the archive directory? Probably not, though once the directory archive is documented it can't be prevented. Provided they respect the archive directory layout, and use a dedicated subdirectory for their internal files, they are free to work with EPIK archive directories as they see appropriate.

What about the CUBE algebra utilities? These generic utilities read and write CUBE-format files, and are not intended to be KOJAK (or EPIK) specific. An example of this, is that the CUBE browser doesn't automatically run EXPERT when provided with an EPILOG trace: in addition to accepting EPIK experiment directories directly, the 'kanal' utility handles both EPILOG and CUBE files, running EXPERT and CUBE as required. The CUBE algebra utilities can be extended to automatically locate CUBE files from within EPIK directories, but the resulting CUBE files are not part of any of the input measurements. Future EPIK algebra utilities might merge or otherwise manipulate EPIK measurement directories.

Should the documented EPILOG (v1) format be changed? Probably not at this stage, though future extensions and modifications may well suggest a new EPILOG v2 format, at which point, disjoint (ASCII-format) definitions and (definition-less) event traces as well as distributed (unmerged) event traces may be included. Until then, externally visible EPILOG traces should contain merged definitions and events, and ELG tools (including EARL/EXPERT) should continue to be backward-compatible in supporting such traces.