Deliverable 2.1
Concise overview of performance metrics and tools

Work Package 2

ProfiT-HPC Consortium

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Abstract

In this deliverable, different performance analysis tools are described and rated based on three different classes: the metrics they deliver, the kind of data collection possible and several operational aspects. This rating was done with the goal of choosing existing tools for our toolkit and the upcoming functional back-end specification for the toolkit in mind.
The tools described in this deliverable range from very simplistic or specialised tools, that handle only one metric but this with very much detail, to complex performance analysis suites, which cover the collection of several metrics and some of which also come with a powerful graphical user interface or viewers. Yet, all tools described in this document are open-source tools, to benefit the roll-out of the final toolkit, where additional costs or necessary licenses might discourage especially smaller compute centres with limited financial resources from the adaption of the toolkit.
After the description and detailed evaluation of the different tools, the results are summed up in two matrices, giving a concise overview over the capabilities of each tool and its usability in the scope of this project.

This document is one of several that will be used as the basis for the design and architectural choices in the final toolkit. It is complemented by a white paper on the Tier-3 infrastructure in Germany (D1.2), the information retrieved from the online survey (D1.1) and the functional specification of the back-end (D2.2).

Contents

Overview of the Evaluation

This document gives a non-exhaustive collection of programs and libraries that can be used to collect performance data of user programs. The different tools are intended to be a basis for work package 3. Different tools, each with its own focus area, are listed and those aspects relevant for our final goals are grouped and evaluated. Besides raw data on performance, some of the tools are capable of detecting performance "patterns", which are generally unique features of some the tools discussed here. For each tool we have gathered generic information and a short list of collected performance metrics and general capabilities. Additionally, an evaluation regarding the features supported (or not supported) by the tool is provided.

We have evaluated each feature and capability of the tool based on the following three classes.

Metrics

The first evaluation measure concerns the presence and extent of performance metrics collected by the tool. The metrics are grouped into more general categories as described below.

General

General CPU resource consumption measurements, including but not limited to timing statistic and source-code annotation are grouped here. This would be the first level of the evaluation , that relates directly to the concept of performance measurement. The output from such metrics is usually the starting point for performance optimisation (e.g., hot spot analysis). Analysing multi-threaded programs in particular (with regards to load-balancing, ideal thread counts and affinity, etc.) is another point of interest to the developers and are grouped in this category.

Hardware Counters

In all modern processors a set of hardware performance counters exists in order to collect micro-architectural events within the CPU. Examples are the number of elapsed cycles, the number of cache misses, and branch mispredictions. Some of these counters indicate to which degree the CPU is optimally used, but correlating these low level performance metrics to the source code itself is a demanding task (considering for instance compiler optimisation). Additionally, extra care has to be taken to interpret mere numbers from the performance counters. In this regard, Ref. [1] is a practical reference in which several performance behaviours and their corresponding signature in the hardware performance metrics are described. Nevertheless, relying only on these metrics (which may be a massive amount of data) typically requires a very good knowledge of the algorithm and the code running on the respective HPC architecture, which may not be possible in an all-inclusive, automated solution.

Memory

Memory utilisation analysis, including usage of physical (e.g., heap) and virtual (pages) memories, are evaluated in this category. The output from these metrics can provide invaluable feedback to the users regarding the overall possibility of improving the performance of the program. For example, a high number of allocating (or deallocating) small memory blocks can be a sign for poor performance (which also may lead to a high cache miss rate). Another example is a too low or a too high memory utilisation (in connection with the total number of paging or page faults), which may be a hint to the possibility for performance improvement.

File System

I/O statistics, such as the amount of data read from or written to the local or networked storage system, fall under this category. For instance, a high number of opened files can be seen as a considerable performance bottleneck within the program.

MPI

Network data (e.g. number of packets, bytes read or written) and MPI performance (communication patterns, load balancing, message size statistics), and MPI correctness are categorised here. A simple metric in this category, such as the percentage of time spent under `MPI_WAIT`, can help to quantify MPI load balancing, which could be the determining factor in scalability.

Data collection

Second, the data collection possibilities of each tool are considered. We have identified two important criteria for our toolkit: whether the underlying tools are capable of sampling or of tracing.

Sampling

Sampling statistical performance data is done in a time-based manner. This method generally introduces a lower performance overhead to the program and less data is produced, but it is not as accurate regarding the fine details of the analysed program, especially in the case of many short functions.

Tracing

Tracing of the user and system calls and functions, is done with the help of instrumentation. This method provides much more accurate data regarding the behaviour of the program under study, but usually imposes a much greater penalty on the program's performance.

Operational aspects

Third and final, operational aspects of the performance analysis done by the tool are evaluated. The rating in this category was done with special focus on the usage within our automated toolkit.

Ease of Use

This category focuses on what has to be done *before* the performance analysis. For example, whether the tool capabilities can be incorporated without interference to the users' workflow (such as the need for re-compilation or re-linking).

Automatability

Complementary, the focus of this category lies on what has to be done *after* obtaining the performance data. For example, whether the output data generated by the tool can be processed without much interaction from users (or developers), and if the reported data are in an accessible format, such as text, XML, or a known database.

Overhead

This category rates the impact of using the tool on the performance of the program (such as increase in the total run-time or the high water mark memory usage).
A low rating in this category is equivalent to a high performance penalty imposed on the program, while a high rating means there is only little overhead to be expected.

Availability

This rating is an a priori assessment of the availability of tool (in a typical HPC environment), based on observations and experience. Example of criteria include whether the tool is included in a standard distribution, another popular tool depends on it, or whether the tool is already deployed on leading HPC centres.

Rating

The following rating is used for the assessment of each criteria regarding the usability of this tool for the planned toolkit.

–	The tool in this respect does not support or is not relevant to our use case.
⋆	The usage of the tool regarding this criteria is possible but not recommended.
⋆⋆	Regarding this measure, using this tool is practical.
⋆⋆⋆	This tool is an excellent candidate to use with respect to our use cases.

GNU/Linux Tools

We have considered the performance tools from the Linux operation system and the GNU project, besides free and open source tools and libraries. Commercial tools have been omitted from this document, since the following tools cover nearly all of the requirements of the subsequent work packages and they might pose an impediment on the successful roll-out and wide-spread usage of the resulting toolkit. Neither of the following tools support MPI or multi-process applications intrinsically. In this regard, the collected data among processes should be manually aggregated. Another point to consider additionally is the support for accelerators. However, since the Intel Knights Landing supports a host operating system, some of the following tools can be or has been ported to the current generation of Intel Xeon Phi processors (albeit not necessarily well tested in production nor scale).

time

`time` is a built-in command in both bash and zsh. It reports the time consumed by the execution of the command (or of the pipeline).

Generic information

- Website: https://www.gnu.org/software/bash/
- License: GNU General Public License v3+
- Version and date: 4.4.5, November 2016

Collected metrics

- Total timing statistics (real time, user CPU time, and system CPU time).

Key general capabilities

- Text output

Evaluation

- General: ⋆
  `time` only reports very basic timing information.
- Hardware Counters: –
- Memory: –
- File System: –
- MPI: –
- Sampling: –
- Tracing: –
- Ease of use: ⋆⋆⋆
  No change is needed to the program.
- Automatability: ⋆⋆⋆
  Usage of the tool and parsing its output is straightforward.

- Overhead: ⋆⋆⋆
  Negligible overhead.
- Availability: ⋆⋆⋆
  Bash is installed on almost all HPC systems.

GNU time

GNU time (located by default under `/usr/bin/time`) runs the specified program command with the given arguments, and after its completion, it writes a message to standard error giving timing statistics about this program run.

Generic information

- Website: https://www.gnu.org/software/time/
- License: GNU General Public License v3+
- Version and date: 1.7.2, January 2015

Collected metrics

- Elapsed real time, user CPU time, and system CPU time,
- Maximum resident set size of the process during its lifetime, average total memory use of the process,
- Number of page faults, swaps and voluntarily and involuntarily context-switched,
- Number of file system I/O by the process,
- Number of socket messages sent/received by the process,
- Number of signals delivered to the process,
- Name, command-line arguments, and the exit status of the command.

Key general capabilities

- Text output
- Bash `time` uses `getrusage()` while GNU time uses `times()`. The former can be more precise because of its microsecond resolution.

Evaluation

- General: ⋆⋆
  GNU time can provide general information regarding common resource usage in an easy and non-intrusive way.
- Hardware Counters: –
- Memory: ⋆
- File System: ⋆
  GNU time can report basic numbers regarding memory consumption and number of files read/written, but not as detailed as one would need to improve the performance.
- MPI: –

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

- Sampling: –
- Tracing: –
- Ease of use: ⋆⋆⋆
  No change is needed to the program.
- Automatability: ⋆⋆⋆
  Usage of the tool and parsing its output is straightforward.
- Overhead: ⋆⋆⋆
  Negligible overhead.
- Availability: ⋆⋆⋆
  GNU time is installed by default on almost all HPC systems.

strace

`strace` intercepts and records the system calls which are called by a process and the signals which are received by a process.

Generic information

- Website: https://strace.io/
- License: BSD
- Version and date: 4.16, February 2017

Collected metrics

- Tracing system calls, signal deliveries, and changes of process state.
- `strace` by itself does not collect data about performance, but its output has to be parsed and interpreted to gather the performance data.

Key general capabilities

- Text output

Evaluation

- General: ⋆
  It is possible to obtain the (relative) time span for each executed system call, and calculate the amount of time needed for such operations.
- Hardware Counters: –
- Memory: ⋆
- File System: ⋆
  Since `strace` reports statistics about system calls, it can be used to collect some information about memory allocation/deallocation and I/O fucntions. However, raw data has to be collected and parsed by the backend.
- MPI: –
- Sampling: –

- Tracing: ⋆⋆
  `strace` traces system function calls.
- Ease of use: ⋆⋆
  Comparatively easy to use and and aggregate the results.
- Automatability: ⋆⋆
  Usage of the tool and parsing its output is straightforward.
- Overhead: ⋆
  `strace` can cause significant and sometimes massive performance overhead to the program, relative to the system call rate it is instrumenting.
- Availability: ⋆ ⋆ ⋆
  `strace` is widely available and can be installed from official binary repositories, but needs `procfs` to be mounted.

ltrace

`ltrace` intercepts and records the dynamic library and system calls within a process, which are called by the executed process and the signals received by that process.

Generic information

- Website: `http://www.ltrace.org/`
- License: GNU General Public License v2
- Version and date: 0.7.3, September 2013

Collected metrics

- Intercepting and recording the dynamic library calls.
- Similar to `strace`, `ltrace` does not collect metrics by itself, but the data should be gathered and parsed.

Key general capabilities

- Text output
- `ltrace` cannot work with multithreaded programs.

Evaluation

- General: ⋆⋆
  It is possible to obtain the (relative) time span for each executed function call, and calculate the amount of time spent for the call.
- Hardware Counters: –
- Memory: ⋆
- File System: ⋆
- MPI: ⋆
  It is possible to trace MPI functions using `ltrace`, but extra care has to be taken with collecting the data among different nodes, since `ltrace` itself does not support MPI directly.

- Sampling: –
- Tracing: ⋆⋆
  `ltrace` traces library calls.
- Ease of use: ⋆⋆
  `ltrace` is reasonably simple to use, but a selection of functions has be done before running the application (in order to reduce the general overhead).
- Automatability: ⋆⋆
  Usage of the tool is straightforward, but a careful analysis has to be done in order to obtain a picture regarding the performance of the application.
- Overhead: ⋆⋆
  `ltrace` suffers from the performance limitations inherent in `ptrace`-based tools. But with careful selection of only functions of interest the overhead can be reduced.
- Availability: ⋆ ⋆ ⋆
  Linux distributions probably already have it available, if not it can be installed using default package managers.

ftrace

`ftrace` (Function Tracer) is a tracing framework built directly into the Linux kernel.

Generic information

- Website: https://www.kernel.org/doc/Documentation/trace/
- License: GNU General Public License v1.2 and v2
- Version and date: same as the Linux kernel

Collected metrics

- `ftrace` can be used for debugging or analysing latencies and performance issues that take place outside of user space.

Key general capabilities

- Text output

Evaluation

- General: ⋆⋆
  The timestamp of the executed functions (from different clocks in the system) can be obtained and used to calculate the time spent in each function.
- Hardware Counters: –
- Memory: ⋆
- File System: ⋆
  `ftrace` can be used to record information related to various function calls and events (such as scheduling events, interrupts, memory-mapped I/O and file system operations) performed while the kernel is running.
- MPI: –

- Sampling: –
- Tracing: ⋆⋆
  `ftrace` is a flexible framework for analysing kernel events.
- Ease of use: ⋆⋆
  Usage of `ftrace` is relatively easy, however, only a selection of functions to trace has to be done before running the application.
- Automatability: ⋆⋆
  Since the main information provided by `ftrace` is of the kernel functions, the textual output has be analysed carefully in order to gain an understanding of the performance bottlenecks within the user application.
- Overhead: ⋆⋆
  Tracing every function incurs a large overhead, but one can filter results only to the desired functions to reduce the general overhead.
- Availability: ⋆ ⋆ ⋆
  `ftrace` is built in the Linux kernel.

perf

`perf` is a command-line profiler tool available from the Linux kernel that abstracts away CPU hardware differences in Linux performance measurements. A broad list of `perf` examples usage can be found in Ref. [3]. In Ref. [4], several useful scripts based on `perf_events` and `ftrace` have been developed.

Generic information

- Website: https://perf.wiki.kernel.org/
- License: GNU General Public License
- Version and date: same as the Linux kernel (version 2.6.31+)

Collected metrics

- CPU performance counters
- Dynamic tracing (tracepoints, kprobes, and uprobes)
- Lightweight profiler

Key general capabilities

- Text output
- Core support for Intel Xeon Phi (Knights Landing, starting from Linux 4.5)

Evaluation

- General: ⋆ ⋆ ⋆
  The `perf` tool can be used to collect profiles on per-thread, per-process and per-CPU basis.
- Hardware Counters: ⋆⋆
  `perf` supports collection of CPU performance monitoring counters.

- Memory: ⋆
- File System: ⋆
- MPI: ⋆
  The profiling data from `perf` can be analysed in order to gather performance data regarding memory usage, file system I/O and network statistics. The data is provided by instrumenting Kernel Tracepoint Events.
- Sampling: ⋆⋆
  `perf` is actively under development. It can instrument hardware counters, and static and dynamic tracepoints.
- Tracing: –
- Ease of use: ⋆⋆
  Using `perf` and parsing its textual output is rather easy and non-interrupting to user programs.
- Automatability: ⋆⋆
  perf can be used in an automated way without any interaction from the user, but the output has to be parsed and analysed in order to provide a summary of performance results.
- Overhead: ⋆
  `perf` can impose drastic penalties on the performance of the application, and often one has to control the sampling rate.
- Availability: ⋆⋆⋆
  `perf` is most probably already present in the HPC systems, if not it can be installed via the package `linux-tools-common` package.

gprof

The GNU profiler `gprof` (part of GNU Binutils) collects and arranges statistics on user programs.

Generic information

- Website: https://sourceware.org/binutils/docs-2.28/gprof/
- License: GNU General Public License v3+
- Version and date: 2.28, March 2017

Collected metrics

- Call graph collecting and printing (unlike `perf`)
- Flat profile with how much time is spent in each function, and how many times that function was called
- Annotated source listing

Key general capabilities

- Text output
- No support for multi-threaded application (profiling only the main thread of application)

Evaluation

- General: ⋆⋆
  `gprof` produces several forms of output: flat profile (time statistics), call graph (of user functions) and annotated source listing with the number of how many each line of program being executed.
- Hardware Counters: –
- Memory: –
- File System: –
- MPI: ⋆
  `gprof` can trace MPI functions as well, but a wrapper is required to collect the data among different processes.
- Sampling: ⋆⋆
  `gprof` uses a hybrid combination of instrumentation and sampling.
- Tracing: –
- Ease of use: ⋆
  In order to use `gprof` a recompile of the source code is required (with additional compiler option '`-pg`').
- Automatability: ⋆⋆
  The textual output of `gprof` is rather customisable and uncomplicated to parse and analyse.
- Overhead: ⋆
  The overhead (mainly caused by instrumentation) from `gprof` can be quite high (reported more than 260% in Ref. [5]).
- Availability: ⋆ ⋆ ⋆
  `gprof` is part of the GNU Binutils, which is most likely present in HPC environments.

procfs

The `proc` file system is a pseudo-file system which provides an interface to kernel data structures about process information.

Generic information

- Website: `https://www.kernel.org/doc/Documentation/filesystems/proc.txt`
- License: GNU General Public License v2
- Version and date: same as the Linux kernel

Collected metrics

Some of the collected metrics for each process include:

- Command line arguments,
- Current and last CPU in which it was executed,
- Memory maps to executables and library files,
- Memory held by this process,

- Process status (also in human readable form),
- Page table,
- Full stack trace report
- Memory consumption and usage (in pages) Values of environment variables,

Key general capabilities

- Text output

Evaluation

- General: ⋆⋆
  `procfs` is mostly used to obtain information regarding the basic process information and general resource consumption of the programs.
- Hardware Counters: –
- Memory: ⋆⋆
- File System: ⋆⋆
  Statistics from the memory usage and file system I/O can be obtained via the `/proc` file system.
- MPI: –
- Sampling: ⋆
  Data from `procfs` has to be read and collected periodically.
- Tracing: –
- Ease of use: ⋆ ⋆ ⋆
  The tool is rather easy to use and not intrusive to the users' workflow.
- Automatability: ⋆ ⋆ ⋆
  The output data from the `procfs` is simple to collect and analyse.
- Overhead: ⋆⋆
  Generally, reading information from `/proc` has low overhead, but a high frequency data readout can cause a noticeable overhead.
- Availability: ⋆ ⋆ ⋆
  `/proc` is part of the Linux kernel and is (mostly) present by default.

sysstat

The sysstat package contains various utilities to monitor system performance and usage activity. sysstat also contains tools that can be scheduled via `cron` or `systemd` to collect performance and activity data:

Generic information

- Website: http://sebastien.godard.pagesperso-orange.fr/
- License: GNU General Public License v2
- Version and date: 11.5.5, February 2017

Collected metrics

- Collective CPU usage and individual CPU statistics,
- Memory and swap space used and available,
- Overall and individual I/O activities of the system and devices,
- Context switch statistics,
- Run queue and load average data,
- Network statistics.

Key general capabilities

- Text, XML, CSV (through `dstat`), and JSON outputs

Evaluation

- General: ⋆⋆
  The package contains tools to collect general information about CPU utilisation and common resources usage.
- Hardware Counters: –
- Memory: ⋆
  The package contains tools to collect basic information about program's memory usage.
- File System: ⋆⋆
  The package contains tools to collect detailed statistics about file system I/O (also network file systems).
- MPI: –
- Sampling: –
- Tracing: –
- Ease of use: ⋆⋆
  The usage of the tools in the package is rather straightforward, and not disruptive to users' workflow.
- Automatability: ⋆⋆
  The collection of the data reported by the tools in the package is simple to analyse, since the output can be also provided in XML and JSON formats as well.
- Overhead: ⋆⋆
  The overhead from the systat package is nearly the same as the `/proc` file system, where the tools read their information from.
- Availability: ⋆ ⋆ ⋆
  The package is simple to install and manage, and likely to be installed by default.

General Tools

This section contains a list of performance related open source tools, that can be relevant to our use cases. The tools are alphabetically sorted.

Darshan

Darshan [6], is a scalable I/O profiling tool, designed to capture an accurate picture of application I/O behaviour.

Generic information

- Website: `http://www.mcs.anl.gov/research/projects/darshan/`
- License: MPICH License [1]
- Version and date: 3.1.3, February 2017

Collected metrics

- I/O access patterns at the POSIX and MPI-IO layers (as well as limited information about HDF5 and PnetCDF)

Key general capabilities

- Portable binary output file format, with tools to generate text and graphical representation of the data.

Evaluation

- General: ⋆⋆
  Darshan reports times spent for the job itself and the (POSIX and MPI) I/O functions.
- Hardware Counters: –
- Memory: –
- File System: ⋆ ⋆ ⋆
  Darshan generate a single log file summarising the I/O activity from the application.
- MPI: ⋆⋆
  Darshan instruments MPI applications to capture MPI-IO file access.
- Sampling: –
- Tracing: ⋆⋆
  Darshan only instruments MPI applications to capture I/O statistics, beside exposing an API that can be used to develop and add new instrumentation modules.
- Ease of use: ⋆⋆
  Statically linked executables must be instrumented at compile time. For dynamically-linked executables, Darshan relies on the `LD_PRELOAD` environment variable to insert instrumentation at run-time.

[1] Usage, reproduction, preparing derivative works, and redistribution to others are granted.

- Automatability: ⋆⋆
  Darshan has its own binary format for the output. A command line utility can be used to obtain a complete, human-readable, text-format dump of all information contained in the log file.
- Overhead: ⋆ ⋆ ⋆
  Initial performance results demonstrate a low overhead of nearly 1% on the Cori system for the IOR benchmark [6].
- Availability: ⋆
  The tool is not included in any default environments or libraries, although it is installed in some of the largest HPC systems worldwide.

gperftools

`gperftools` (originally Google Performance Tools) is a collection of a high-performance multi-threaded `malloc()` implementation, plus several performance analysis tools such as a heap profiler.

Generic information

- Website: https://github.com/gperftools/gperftools
- License: 3-Clause BSD Licence
- Version and date: 2.5, March 2016

Collected metrics

- Heap memory checker and profiler, memory leak finder
- CPU profiler

Key general capabilities

- Binary (but open) format, and text output

Evaluation

- General: ⋆⋆
  The CPU profiler collects statistics about the time spent in functions.
- Hardware Counters: –
- Memory: ⋆ ⋆ ⋆
  The heap profiler reports what is in the program heap at any given time and locates memory leaks. It can also be used to find the places that perform large number of allocations.
- File System: –
- MPI: –
- Sampling: ⋆⋆
  The tool's CPU profiler reports the statistical data of samples in each function.
- Tracing: –

- Ease of use: ⋆⋆
  The code has to be re-linked to this library. The usage of `LD_PRELOAD` is supported but not recommended.
- Automatability: ⋆⋆
  The textual output of the tool is easy to parse and analyse.
- Overhead: ⋆⋆⋆
  The code may even gain some extra performance by linking to `gperftools`, since the implemented `malloc()` function is faster than the standard library (beside being multi-threaded).
- Availability: ⋆
  `gperftools` is developed and used by Google. It probably is not installed on a default HPC system.

HPCToolkit

HPCToolkit is a multi-platform suite of tools for tracing, profiling and analysing program performances.

Generic information

- Website: `http://hpctoolkit.org/`
- License: 3-Clause BSD License
- Version and date: 2016.12, December 2016

Collected metrics

- Support for serial, threaded (e.g. pthreads and OpenMP), MPI, and hybrid (MPI+threads) programs.

Key general capabilities

- Support for Intel Knights Landing
- Ability to export performance data to CSV and XML formats

Evaluation

- General: ⋆⋆⋆
  Wall-clock, real and CPU times are reported by HPCToolkit.
- Hardware Counters: ⋆⋆
  Support for hardware performance counters is provided using PAPI.
- Memory: ⋆⋆
  Memory allocation, deallocation and possible leaks are reported.
- File System: ⋆⋆
  I/O bytes read and written is reported by sampling.
- MPI: ⋆⋆
  HPCToolkit's measurement tools collect data on each process and thread of an MPI program, but no specific analysis is done regarding message passing date.

- Sampling: ⋆⋆
  HPCToolkit supports statistical gathering of the performance metrics (e.g. hardware counters and thread call stacks).
- Tracing: ⋆
  HPCToolkit favours the use of *asynchronous sampling* to measure and attribute performance metrics.
- Ease of use: ⋆⋆
  For dynamically linked programs, HPCToolkit requires no changes done to the program source and no change to the build procedure. For statically linked executables, it require no source-code modifications, but a re-link (using a provided command) is needed.
- Automatability: ⋆
  The output format of HPCToolkit is supposed to be visualised by the tool itself. Although the raw data can be exported to CSV and XML formats, this has to be done using the GUI tool.
- Overhead: ⋆⋆
  Since HPCToolkit uses sampling, measurement has low overhead (1-3%) and scales to large parallel systems (according to the HPCTooklit manual).
- Availability: ⋆⋆
  The tool is not likely to be installed under a default HPC environment, however, it has been deployed on several DOE supercomputing systems, and Bull Extreme Computing distributes HPCToolkit as part of its bullx SuperComputer Suite development environment.

IPM

IPM (Integrated Performance Monitoring) is a extremely low-overhead, scalable, and portable profiling infrastructure for a parallel program.

Generic information

- Websites: `http://ipm-hpc.sourceforge.net/`, `https://github.com/nerscadmin/IPM`
- License: GNU Library or Lesser General Public License version 2.0
- Version and date: 0.983, August 2010 (Alpha release 2.0.6, July 2016)

Collected metrics

- MPI: Communication topology and statistics for each MPI call and buffer size
- HPM: FLOPs and such via PAPI on-chip event counters
- OpenMP: thread level details about load imbalance (alpha release)
- Memory: memory high watermark, `getrusage`, `sbrk`
- Switch: Communication volume and packet loss
- File I/O: Data written to and read from disk
- GPU: Memory copied in/out of the GPU and time in kernels (alpha release)
- Power: Joules of energy consumed by the app (alpha release)

Key general capabilities

- Text and XML output
- Modular design

Evaluation

- General: ⋆⋆⋆
  IPM measures core timings of the application.
- Hardware Counters: ⋆⋆
  Hardware performance counter collection is done via PAPI.
- Memory: ⋆⋆
  IPM gathers information regarding program's memory usage.
- File System: ⋆⋆
  IPM gathers information about the data written to and read from the disk.
- MPI: ⋆⋆
  IPM collects communication topology and statistics for each MPI call.
- Sampling: ⋆⋆
  IPM has a modular structure to measure different metrics.
- Tracing: –
- Ease of use: ⋆⋆
  In the static usage the users code needs to be re-linked. In the dynamic case load the IPM libraries using `LD_PRELOAD` is supported.
- Automatability: ⋆⋆
  IPM produces a summary as text and an XML file that can be used to generate a graphical web page.
- Overhead: ⋆⋆⋆
  IPM maintaining a small fixed memory footprint and using minimal amounts of CPU, with having low-overhead as a design goal.
- Availability: ⋆
  The tool is not included in any default environments or libraries.

LIKWID

LIKWID [7] is a set of lightweight command-line tools for performance oriented programmers.

Generic information

- Website: https://github.com/RRZE-HPC/likwid
- License: GNU General Public License v3
- Version and date: 4.2.0, December 2016

Collected metrics

- Hardware performance counters on Intel and AMD processors, including a monitoring agent,
- RAPL Energy information,
- Pinning threaded application (pthreads as well as Intel and GCC OpenMP) to dedicated processors,
- Micro benchmarking platform,
- Wrapper to start MPI and Hybrid MPI/OpenMP applications (Supports Intel MPI, Open-MPI and MPICH),
- CPU frequency control.

Key general capabilities

- Text output
- Support for Intel Knights Landing

Evaluation

- General: ⋆⋆
  The benchmarking tool of LIKWID enable rapid prototyping of multi-threaded benchmarking kernels.
- Hardware Counters: ⋆⋆⋆
  LIKWID byreads Linux kernel and reads the hardware performance counters directly from the MSR registers.
- Memory: –
- File System: –
- MPI: –
- Sampling: –
- Tracing: –
- Ease of use: ⋆⋆
  LIKWID can be used to perform threaded execution and pinning, data allocation and placement, time measurement and result presentation.
- Automatability: ⋆⋆
  LIKWID output is in text, and includes a system-wide monitoring agent to measure hardware performance counters.
- Overhead: ⋆⋆
  LIKWID has generally low overhead. Details of overhead measurements has been done in Ref. [8].
- Availability: ⋆⋆
  LIKWID is installed on some of top tier German HPC systems.

MAQAO

MAQAO (Modular Assembly Quality Analyzer and Optimizer) is a modular static and dynamic binary code analyser and optimising tool, which provides static analysis, instrumentation and assembly rewriting capabilities in a single integrated framework. One included module is the MAQAO lightweight profiler (LProf) that allows developers to easily profile an application to detect hot functions and loops using either sampling (default mode) or with help of instrumentation.

Generic information

- Website: http://www.maqao.org/
- License: GNU Lesser General Public License v3+
- Version and date: 2.2.0, February 2016

Collected metrics

- Static and dynamic code analyser and profiler

Key general capabilities

- Text and HTML output

Evaluation

- General: ⋆⋆⋆
  MAQAO gives lightweight localisation of application hot spots, with a loop-centric approach.
- Hardware Counters: ⋆⋆
  MAQAO measures hardware counters in sampling mode.
- Memory: ⋆⋆
- File System: ⋆⋆
  MAQAO supports time categorisation within the binary, and supports I/O, string, memory, and external library operations.
- MPI: ⋆⋆
  MAQAO characterises MPI functions within the program and provides timing, hits (number of calls), as well as message size statistics.
- Sampling: ⋆⋆
  MAQAO has two measurement mode, default being sampling and non-intrusive with low overhead.
- Tracing: ⋆⋆
  Instrumentation is also supported by MAQAO and is done using binary rewriting.
- Ease of use: ⋆⋆⋆
  No need for re-compile or re-link of the program is required. Furthermore, MAQAO is is scriptable in Lua, which allows customised static and dynamic analysis of the user programs.
- Automatability: ⋆⋆⋆
  MAQAO generates text and HTML reports (including advice for further optimisation) as the output for the analysis.

- Overhead: ⋆⋆
  MAQAO profiling using sampling has relatively very low overhead. Instrumentation introduces higher overhead for more precise results.
- Availability: ⋆
  The tool is not likely to be installed in a default installation, but the official binary packages are provided via their website.

memP

memP is a parallel, lightweight heap profiling library designed to collect basic memory consumption information, and is based on the mpiP MPI profiling tool.

Generic information

- Website: `http://memp.sourceforge.net/`
- License: 3-Clause BSD Licence
- Version and date: 1.0.3, April 2010

Collected metrics

- Heap memory profiling

Key general capabilities

- Text and XML output

Evaluation

- General: –
- Hardware Counters: –
- Memory: ⋆ ⋆ ⋆
  The goal of memP is identify the heap allocation that causes a task to reach its memory in use high water mark for each task in a parallel job.
- File System: –
- MPI: –
- Sampling: –
- Tracing: –
- Ease of use: ⋆⋆
  Using memP requires either using a special run-time script or re-linking with the memP library.
- Automatability: ⋆⋆
  The XML output of memP is intended to be examined with another tool, Tool Gear [9].
- Overhead: ⋆⋆
  In Ref. [10] the run-time slowdown for several applications has been measured (from none to 80x overhead).
- Availability: ⋆
  memP is available on LLNL Linux systems.

MPE

The MPE extensions to the MPI library provide a number of useful facilities for MPI programmers, including several profiling libraries to collect information, postmortem visualisation of log files and real-time animation.

Generic information

- Website: http://www.mcs.anl.gov/research/projects/perfvis/
- License: MPICH License
- Version and date: 2-2.4.9b, August 2015

Collected metrics

- A set of thread-safe profiling libraries to collect information about the behaviour of MPI programs.

Key general capabilities

- Graphical representations MPI profiling events
- an MPI collective and datatype checking library
- Compiler wrappers (`mpecc` and `mpefc`) for compiling/linking with the related profiled libraries
- A profiling wrapper generator for MPI interface

Evaluation

- General: –
- Hardware Counters: –
- Memory: –
- File System: –
- MPI: ⋆⋆⋆
  The latest version of MPE (MPE2) has an extensive set of tools to profile MPI libraries, and is shown to work with many open and commercial MPI libraries.
- Sampling: –
- Tracing: ⋆⋆
  The MPI profiling interface provides a convenient way to add performance analysis tools to any MPI implementation.
- Ease of use: ⋆⋆
  A re-link of the program in order to link to the tracing libraries is required. The usage of `LD_PRELOAD` is not discussed in the original documentation.
- Automatability: ⋆
  The output log files are aimed for a graphical representation directly, although converter tools exists for the textual output of the log.

- Overhead: ⋆⋆
  The default log file format (CLOG2) is a low overhead logging format, which contains a simple collection of single time-stamp events.
- Availability: ⋆⋆
  MPE has been seamlessly integrated into the MPICH distribution.

mpiP

mpiP is a lightweight and scalable profiling library for MPI applications.

Generic information

- Website: `http://mpip.sourceforge.net/`
- License: BSD
- Version and date: 3.4.1, March 2014

Collected metrics

- Statistical information about MPI functions

Key general capabilities

- Text output
- High scalability (tested on a variety of C/C++ /Fortran applications from 2 to 262144 processes)

Evaluation

- General: –
- Hardware Counters: –
- Memory: –
- File System: –
- MPI: ⋆ ⋆ ⋆
  mpiP collects statistical data about almost all MPI functions.
- Sampling: ⋆⋆
  mpiP is a profiler only for MPI functions, and its usage is not sufficient to reveal other details of the program.
- Tracing: –
- Ease of use: ⋆⋆
  mpiP is a link-time library and needs re-linking of the application. Usage of `LD_PRELOAD` is tested in production, such as in Ref. [11].
- Automatability: ⋆⋆
  The output files are in text format and uncomplicated to parse and analyse.
- Overhead: ⋆ ⋆ ⋆
  A design goal of mpiP is being lightweight, generating considerably less overhead and much less data than tracing tools. For a study using mpiP, see Ref. [12].

- Availability: ⋆⋆
  The tool is not included in any default environments or libraries, however it is also distributed via the UNITE (UNiform Integrated Tool Environment) installer.

MUST

MUST is a tool designed to detects usage errors of the MPI functions.

Generic information

- Website: https://doc.itc.rwth-aachen.de/display/CCP/Project+MUST
- License: BSD
- Version and date: 1.5.0, July 2016

Collected metrics

- MPI correctness checker

Key general capabilities

- HTML output

Evaluation

- General: –
- Hardware Counters: –
- Memory: –
- File System: –
- MPI: ⋆⋆
  MUST reports errors in the usage of the MPI functions, but does not provide any information on the performance of the application. However, it detects possible deadlocks or leaking MPI resources, that can have a sizeable performance cost.
- Sampling: –
- Tracing: ⋆⋆
  MUST can utilise Dyninst to collect the call stack information.
- Ease of use: ⋆⋆
  Usage of MUST is very simple, however it adds an additional MPI task for the correctness checking.
- Automatability: ⋆⋆
  MUST reports the results in an HTML file, containing all detected issues and where the error has been occurred.
- Overhead: ⋆⋆
  MUST has several modes of operation to adapt its overhead to the target use case. According to the MUST manual, the distributed mode is tested with 16,384 processes.
- Availability: ⋆⋆
  MUST is not likely to be included in default HPC environments, however it is also distributed via the UNITE (UNiform Integrated Tool Environment) installer.

ompP

ompP is a profiling tool for OpenMP applications written in C/C++ or Fortran.

Generic information

- Website: http://www.ompp-tool.com/
- License: GNU General Public License
- Version and date: 0.8.5, August 2014

Collected metrics

- OpenMP profiler

Key general capabilities

- Text and CSV output

Evaluation

- General: ⋆⋆
  ompP offers insight regarding parallel performance data of the OpenMP programs.
- Hardware Counters: ⋆
  ompP supports the measurement of hardware performance counters using PAPI, but it can record up to only four counters simultaneously.
- Memory: –
- File System: –
- MPI: –
- Sampling: –
- Tracing: ⋆
  ompP only traces OpenMP calls.
- Ease of use: ⋆
  A re-compile is needed to use this tool.
- Automatability: ⋆⋆
  The text report output of the tool is uncomplicated, supported by the possibility of CSV output.
- Overhead: ⋆⋆
  The tool itself performs an OpenMP overhead analysis, targeted to examine the parallel coverage within the program.
- Availability: ⋆
  The tool is not likely to be installed in a default installation.

OProfile

OProfile is an open source, system-wide statistical profiling tool for Linux.

Generic information

- Website: http://oprofile.sourceforge.net/
- License: GNU General Public License v2
- Version and date: 1.1.0, August 2015

Collected metrics

- Statistical, system-wide profiler
- `gprof`-style call-graph profiling data (only on x86 and ARM architectures)

Key general capabilities

- Unobtrusiveness to users' workflow
- Text and XMK outputs

Evaluation

- General: ⋆⋆
  Basic timing of the user functions can be done using Oprofile.
- Hardware Counters: ⋆⋆
  Oprofile enables collection of various low-level data from CPU counters, with association to particular sections of code.
- Memory: ⋆⋆
- File System: ⋆⋆
- MPI: ⋆⋆
  Since Oprofile is a general profiler, one can obtain the statistics about the memory, I/O, and MPI function calls.
- Sampling: ⋆⋆
  OProfile can be used to count native hardware events occurring in either a given application, a set of processes or threads, a subset of active system processors, or the entire system.
- Tracing: –
- Ease of use: ⋆ ⋆ ⋆
  Oprofile needs no re-compile or wrapper libraries, and its usage is uncomplicated.
- Automatability: ⋆⋆
  It is possible to obtain the performance report output in XML format.
- Overhead: ⋆⋆
  Oprofile is low overhead with a typical overhead of 1-8%, dependent on sampling frequency and workload.
- Availability: ⋆⋆
  Oprofile is a common used profiler in Linux systems.

Open|SpeedShop

Open|SpeedShop is an open source multi platform Linux performance tool which is targeted to support performance analysis of applications running on both single node and large scale HPC systems.

Generic information

- Website: https://openspeedshop.org/
- License: GNU Lesser General Public License
- Version and date: 2.3, February 2017

Collected metrics

- Program counter sampling,
- Support for call stack and path analysis,
- Hardware performance counter sampling,
- MPI lightweight profiling and tracing,
- I/O lightweight profiling and tracing,
- Floating point exception analysis,
- Memory trace analysis,
- OpenMP profiling and analysis,
- POSIX thread trace analysis,
- NVIDIA CUDA Tracing and Analysis.

Key general capabilities

- Text output as well as SQLite database file
- Modular design
- GPU support (Nvidia CUDA)

Evaluation

- General: ⋆⋆⋆
  Open|SpeedShop can report general timing data in the user application.
- Hardware Counters: ⋆⋆
  Support for hardware performance counters is provided using PAPI.
- Memory: ⋆⋆
  Memory statistics and possible leaks can be tracked using Open|SpeedShop.
- File System: ⋆⋆
  Open|SpeedShop accumulates wall time duration of I/O system calls.
- MPI: ⋆⋆
  Time spent and the number of calls for the MPI functions is recorded by Open|SpeedShop.

- Sampling: ⋆⋆
  Periodic sampling of the user application provides a low-overhead view of the time spent in the functions.
- Tracing: ⋆⋆
  I/O, MPI, memory, and POSIX threads functions can be traced to record performance information.
- Ease of use: ⋆⋆
  Usage of the experiments in Open|SpeedShop is straightforward, and users can run or view single or multiple experiments simultaneously.
- Automatability: ⋆⋆⋆
  Open|SpeedShop offers a command-line interface (CLI), a graphical user interface (GUI) and a python scripting API user interface. Furthermore, the log files data items are in a SQLite database.
- Overhead: ⋆⋆
  Generally the overhead of the performance analysis is dependent on the type of the experiment done, but some experiments are specifically designed to lower the performance penalty.
- Availability: ⋆
  The tool is not likely to be included in any default environments.

PAPI

PAPI (Performance API) is a portable interface to hardware performance counters on modern microprocessors. A list of various CPU support for `perf_event` and PAPI can be find in Ref. [13]. In Ref. [14], the overhead of self-monitoring performance counter measurements on the Linux `perf_event` interface (such as used by PAPI) has been investigated.

Generic information

- Website: http://icl.utk.edu/papi/
- License: 3-Clause BSD License
- Version and date: 5.5.1, November 2016

Collected metrics

- Hardware performance counters

Key general capabilities

- Support for Intel Knights Landing
- Support for Nvidia GPUs (CUDA)
- Measurements performed per thread

Evaluation

- General: –
- Hardware Counters: ⋆⋆⋆
  PAPI is the standard interface to access hardware performance counters.
- Memory: –
- File System: –
- MPI: –
- Sampling: –
- Tracing: –
- Ease of use: ⋆⋆
  There are two methods of using the hardware performance counters. First by aggregating (direct) the value of the counters before and after each call, and second by statistical profiling (indirect) via generating an interrupt when a performance counter reaches a preset value.
- Automatability: ⋆⋆⋆
  PAPI provides a portable way to access the hardware counters.
- Overhead: ⋆⋆
  Aggregate Counts provide low-overhead (but low-detailed), while a sampled execution lead to a variable overhead (with medium details) based on the sampling rate.
- Availability: ⋆⋆⋆
  PAPI is implemented on a wide variety of architectures and operating systems and is probably installed in an HPC environment.

Paradyn

Paradyn is a performance measurement tool to enable dynamic instrumentation of unmodified executables of parallel and distributed programs, using the Dyninst API.

Generic information

- Website: `http://www.paradyn.org/`, `http://www.dyninst.org/`
- License: GNU Lesser General Public License v2.1
- Version and date: Dyninst 9.3, December 2016

Collected metrics

- Dynamic binary instrumentation

Key general capabilities

- Support for Intel Knights Landing

Evaluation

- General: –
- Hardware Counters: –
- Memory: –
- File System: –
- MPI: –
- Sampling: –
- Tracing: ⋆⋆⋆
  Dyninst enables efficient obtaining of the performance profiles of unmodified executables.
- Ease of use: ⋆
  Dyninst is just the framework for parsing and analysing a binary executable. Several other profilers rely on Dyninst for instrumenting the user code.
- Automatability: ⋆⋆
  The performance data of the instrumented code can be collected in on of its three execution modes: Rewriter Mode (instrumentation of existing binaries), Attach Mode (for running processes), and Create Mode (for new processes).
- Overhead: ⋆⋆
  Many other performance tools that offer binary instrumentation (such as TAU) use Dyninst as the underlying instrumentation framework.
- Availability: ⋆⋆
  Other tools depend on Dyninst as an external library, and it can be already installed on the system.

PMPI

PMPI is the profiling interface to the MPI library [15]. The MPI implementation allows selective replacement of MPI routines to their counterpart PMPI routines (hence selectable at link time), allowing adding additional behaviour to the normal MPI function call. A low-overhead library on top of PMPI has been developed in Refs. [16, 17].

Generic information

- Website: `http://www.mpi-forum.org`
- Version and date: MPI version 3.1, June 2015

Collected metrics

- Performance measurement for MPI functions

Key general capabilities

- Standard interface for profiling MPI functions

Evaluation

- General: –
- Hardware Counters: –
- Memory: –
- File System: –
- MPI: ⋆⋆⋆
  PMPI is included in the MPI standard and can be used for all MPI functions.
- Sampling: –
- Tracing: ⋆⋆
  PMPI is only targeted for MPI functions. Other tracing experiments has to be done in parallel in order to obtain other performance data.
- Ease of use: ⋆⋆
  The profiling interface can be selected by pre-loading a wrapper library to exchange chosen MPI functions to their PMPI counterpart.
- Automatability: ⋆⋆
  The above-mentioned wrapper library has to aggregate the performance data collected from each PMPI call.
- Overhead: ⋆⋆⋆
  Basic PMPI profiling has been shown to have less than 1% of overhead [18]
- Availability: ⋆⋆⋆
  Almost all MPI implementations support standard profiling interface.

Score-P

Score-P (Scalable Performance Measurement Infrastructure for Parallel Codes [19]) is a software system that provides a measurement infrastructure for profiling, event trace recording, and online analysis of HPC applications.

Generic information

- Website: `http://www.score-p.org/`
- License: 3-Clause BSD License
- Version and date: 3.0

Collected metrics

- Application instrumentation
- MPI performance measurements

Key general capabilities

- Support for a number of analysis tools. Currently, it works with Periscope, Scalasca, Vampir, and Tau.
- Support for CUDA
- Support for Intel Knights Landing, but with a very costly instrumentation
- Support for OpenCL and OpenACC
- Text output and support for other open tracing formats

Evaluation

- General: ⋆⋆⋆
  Score-P records resource usage statistics for each thread.
- Hardware Counters: ⋆⋆
  Score-P provides the possibility to query hardware performance counters and include these metrics into the trace and/or profile.
- Memory: ⋆⋆
  Score-P records memory usage of the program (enabled by default).
- File System: ⋆⋆
  Only MPI-IO function calls are intercepted by Score-P.
- MPI: ⋆⋆
  The MPI adapter of Score-P supports the tracing of most of MPI functions.
- Sampling: –
- Tracing: ⋆⋆⋆
  Score-P is an instrumentation framework supporting compiler, source-code, and user instrumentation.
- Ease of use: ⋆
  Score-P needs user programs to be recompiled, therefore it limits its usage in an automatic, system-wide manner.
- Automatability: ⋆⋆⋆
  The collected performance data can be stored in the OTF2, CUBE4, or TAU snapshot formats or queried via the on-line access interface.
- Overhead: ⋆⋆
  Score-P has generally low overhead. Some measurements has been done in Ref. [19], and results were shown to be below 4%.
- Availability: ⋆⋆
  Score-P is a well-known profiler and likely to be present (or easily installed) in a HPC system.

Slurm

Slurm Workload Manager provides an API for job profiling by periodic sampling of each tasks CPU use, memory use, power consumption, network and file system use.

Generic information

- Website: https://slurm.schedmd.com/
- License: GNU General Public License v2
- Version and date: 17.02, March 2017

Collected metrics

- Energy consumption data
- I/O data from a network interface (InfiniBand)
- I/O data from parallel file systems (Lustre)
- Task performance data, such as local disk I/O, CPU consumption, and memory usage

Key general capabilities

- Output in HDF5 format

Evaluation

- General: ⋆⋆
  CPU time, utilisation and frequency and the time of the each sample is recorded.
- Hardware Counters: –
- Memory: ⋆⋆
  Memory usage (resident set and virtual memory sizes, number of pages) per task is recorded.
- File System: ⋆⋆
  File system data (read/write bytes) is collected. At the time of writing only Lustre file system is supported.
- MPI: ⋆⋆
  Network (InfiniBand) data (packets in/out, bytes read/write) is collected.
- Sampling: ⋆⋆
  Slurm can periodically sample various performance data either collected by itself, the operating system, or component software.
- Tracing: –
- Ease of use: ⋆⋆⋆
  No special change is required to the code, and users can record the performance data easily, using the `-profile` option by job submission.
- Automatability: ⋆⋆⋆
  Detailed time-series data is stored in an HDF5 file for the job.
- Overhead: ⋆⋆
  As the primary factor for overhead, the frequency of gathering performance data can be overridden (and different sample rates can be specified for each data type). But generally using moderate sampling values no additional overhead is expected.
- Availability: ⋆⋆⋆
  Slurm is widely used job scheduler by many of the world's supercomputers and computer clusters.

TAU

TAU (Tuning and Analysis Utilities) is a portable profiling and tracing toolkit for performance analysis of parallel programs.

Generic information

- Website: http://tau.uoregon.edu/
- License: BSD
- Version and date: 2.26.1, March 2017

Collected metrics

- MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
- POSIX threads, OpenMP, OMPT interface, hybrid, other thread models
- GPU, CUDA, OpenCL, OpenACC

Key general capabilities

- OpenACC and CUDA support

Evaluation

- General: ⋆⋆⋆
  TAU provides statistics about timings of the program.
- Hardware Counters: ⋆⋆
  Hardware performance counters are supported via PAPI.
- Memory: ⋆⋆
  TAU collects information about application's memory usage patterns and possible leaks/
- File System: ⋆⋆
  TAU records information regarding I/O characteristics of the code, such as bandwidth, peak read and write, and total volume.
- MPI: ⋆⋆⋆
  TAU supports all MPI implementations.
- Sampling: ⋆⋆
  TAU covers event based sampling without the need to instrument the binary.
- Tracing: ⋆⋆
  TAU has two methods for source based instrumentation, either from using PDT (source-code parser) or compiler-based. The former method is usually the prefered one. Also, TAU supports binary instrumentation with either MAQAO, DynInstAPI or PEBiL, the choice of instrumentor being configuration-dependent.
- Ease of use: ⋆⋆
  TAU provides three methods to track the performance of the application: Interposition, Compiler, and Source, with no need for re-compilation from the former. Dynamic instrumentation is achieved through library pre-loading.

- Automatability: ⋆⋆⋆
  TAUdb (TAU Database) provides an API/framework to manage and analyse performance data in a database such as MySQL or PostgreSQL.
- Overhead: ⋆⋆
  Using TAU has a performance overhead (amount depending on the type of analysis). However, TAU can optionally throttle short running functions in an effort to reduce the amount of overhead associated with profiles of such functions.
- Availability: ⋆⋆
  TAU is present in several world class HPC systems. It is also distributed via the UNITE (UNiform Integrated Tool Environment) installer.

Valgrind

Valgrind is an instrumentation framework for profiling and building dynamic analysis tools, mostly known for its ability for memory debugging and memory leak detection.

Generic information

- Website: http://valgrind.org/
- License: GNU General Public License v2
- Version and date: 3.12.0, October 2016

Collected metrics

- Memory usage, debugging, and leak detection
- Traceback analysis and callgraph analyser
- Cache and heap profiler
- Performance simulator (from a small sample test)

Key general capabilities

- Text and partial XML output
- Support for multi-threaded programs (although the overhead might be as well multiplied)

Evaluation

- General: –
- Hardware Counters: –
- Memory: ⋆⋆⋆
  Valgrind and its depending tools provide much helpful and detailed information about memory usage and management within the program.
- File System: –
- MPI: –
- Sampling: –

- Tracing: ⋆⋆
  Valgrind is in essence a virtual machine which uses just-in-time compilation technique tp run and analyse the code.
- Ease of use: ⋆⋆
  Usage of Valgrind is simple, although in order to get the more exact results no optimisation should be used for the program. In case of statically linked executables, most Valgrind tools will only work well if they are able to replace certain functions (such as `malloc`) with their own version.
- Automatability: ⋆⋆
  Textual output of Valgrind (and its tools) is in text format, targeted to be read by the users directly.
- Overhead: ⋆
  Although Valgrind memory check tool (Massif) provides an extensive analysis of the program, it comes at the cost of a large performance penalty (even up to 100 times slower).
- Availability: ⋆⋆
  Since Valgrind is a popular tool, it is installed already on several HPC systems. Callgrind and KCachegrind are distributed via the UNITE installer.

Summary

After having given a thorough insight on all aspects relevant to this project for all tools, tables 1 and 2 summarise the evaluation given in the previous sections in a concise manner. As before, the rating is visualised with the legend from section 1.4, where more stars symbolise a better rating in the according category.

	General	HW Counters	Memory	File System	MPI	Sampling	Tracing	Ease of Use	Automatability	Overhead	Availability
`time`	⋆	–	–	–	–	–	–	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
GNU `time`	⋆⋆	–	⋆	⋆	–	–	–	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
`strace`	⋆	–	⋆	⋆	–	–	⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆⋆
`ltrace`	⋆⋆	–	⋆	⋆	⋆	–	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
`ftrace`	⋆⋆	–	⋆	⋆	–	–	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
`perf`	⋆⋆⋆	⋆⋆	⋆	⋆	⋆	⋆⋆	–	⋆⋆	⋆⋆	⋆	⋆⋆⋆
`gprof`	⋆⋆	–	–	–	⋆	⋆⋆	–	⋆	⋆⋆	⋆	⋆⋆⋆
`procfs`	⋆⋆	–	⋆⋆	⋆⋆	–	⋆	–	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆
`sysstat`	⋆⋆	–	⋆	⋆⋆	–	–	–	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆

Table 1: Summary of GNU/Linux performance measurements tools.

	General	HW Counters	Memory	File System	MPI	Sampling	Tracing	Ease of Use	Automatability	Overhead	Availability
Darshan	★★	–	–	★★★	★★	–	★★	★★	★★	★★★	★
gperftools	★★	–	★★★	–	–	★★	–	★★	★★	★★★	★
HPCToolkit	★★★	★★	★★	★★	★★	★★	★	★★	★	★★	★★
IPM	★★★	★★	★★	★★	★★	★★	–	★★	★★	★★★	★
LIKWID	★★	★★★	–	–	–	–	–	★★	★★	★★	★★
MAQAO	★★★	★★	★★	★★	★★★	★★	★★	★★★	★★★	★★	★
memP	–	–	★★★	–	–	–	–	★★	★★	★★	★
MPE	–	–	–	–	★★★	–	★★	★★	★	★★	★★
mpiP	–	–	–	–	★★★	★★	–	★★	★★	★★★	★★
MUST	–	–	–	–	★★	–	★★	★★	★★	★★	★★
ompP	★★	★	–	–	–	–	★	★	★★	★★	★
OProfile	★★	★★	★★	★★	★★	★★	–	★★★	★★	★★	★★
O|SS	★★★	★★	★★	★★	★★	★★	★★	★★	★★★	★★	★
PAPI	–	★★★	–	–	–	–	–	★★	★★★	★★	★★★
Paradyn	–	–	–	–	–	–	★★★	★	★★	★★	★★
PMPI	–	–	–	–	★★★	–	★★	★★	★★	★★★	★★★
Score-P	★★★	★★	★★	★★	★★	–	★★★	★	★★★	★★	★★
Slurm	★★	–	★★	★★	★★	★★	–	★★★	★★★	★★	★★★
TAU	★★★	★★	★★	★★	★★★	★★	★★	★★	★★★	★★	★★
Valgrind	–	–	★★★	–	–	–	★★	★★	★★	★	★★

Table 2: Summary of free and open-source performance measurements tools and libraries.

References

[1] Jan Treibig, Georg Hager, and Gerhard Wellein (2012), Best practices for HPM-assisted performance engineering on modern multicore processors. http://arxiv.org/pdf/1206.3738.pdf

[2] https://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:SandyFlops

[3] http://www.brendangregg.com/perf.html

[4] https://github.com/brendangregg/perf-tools

[5] Nathan Froyd, John Mellor-Crummey, and Rob Fowler. 2005. "Low-overhead call path profiling of unmodified, optimized code". In Proceedings of the 19th annual international conference on Supercomputing (ICS '05). ACM, New York, NY, USA, 81-90. DOI=http://dx.doi.org/10.1145/1088149.1088161

[6] https://www.mcs.anl.gov/research/projects/darshan/docs/DXT-overhead.pdf

[7] J. Treibig, G. Hager and G. Wellein: "LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments." Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, San Diego CA, September 13, 2010. DOI: 10.1109/ICPPW.2010.38

[8] Roehl, T.; Treibig, J.; Hager, G.; Wellein, G.: "Overhead Analysis of Performance Counter Measurements," 43rd International Conference on Parallel Processing Workshops (ICCPW), 2014, pp.176,185, 9-12 Sept. 2014, DOI: 10.1109/ICPPW.2014.34

[9] Tool Gear, https://computation.llnl.gov/casc/tool_gear/

[10] O.F.J. Perks, D.A. Beckingsale, S.D. Hammond, I. Miller, J.A. Herdman, A. Vadgama, A.H. Bhalerao, L. He, S.A. Jarvis; "Towards Automated Memory Model Generation Via Event Tracing," Comput J 2013; 56 (2): 156-174. DOI: 10.1093/comjnl/bxs051

[11] DKRZ User Portal Documentation, Lightweight MPI analysis https://www.dkrz.de/Nutzerportal-en/doku/mistral/program-analysis/copy_of_getrusage

[12] Vetter, J.S. and M.O. McCracken, "Statistical Scalability Analysis of Communication Operations in Distributed Applications", Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP), 2001.

[13] http://web.eece.maine.edu/~vweaver/projects/perf_events/support.html

[14] V. M. Weaver, "Self-monitoring overhead of the Linux perf_event performance counter interface," 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, pp. 102-111. DOI: 10.1109/ISPASS.2015.7095789

[15] MPI: Message Passing Interface Forum (2015), A Message-Passing Interface Standard Version 3.1.

[16] M. Schulz and B. R. De Supinski, "A Flexible and Dynamic Infrastructure for MPI Tool Interoperability," 2006 International Conference on Parallel Processing (ICPP'06), Columbus, OH, 2006, pp. 193-202. DOI: 10.1109/ICPP.2006.6

[17] https://github.com/LLNL/PnMPI

[18] Zoltán Péter Szebenyi, "Capturing Parallel Performance Dynamics", Volume 12 of Schriften des Forschungszentrums Jülich / IAS series: IAS series, Forschungszentrum Jülich, Forschungszentrum Jülich, 2012.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

[19] Knüpfer A. et al.; “Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir,” In: Brunst H., Müler M., Nagel W., Resch M. (eds) Tools for High Performance Computing. Springer, Berlin, Heidelberg, 2012, DOI: 10.1007/978-3-642-31476-6_7

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1