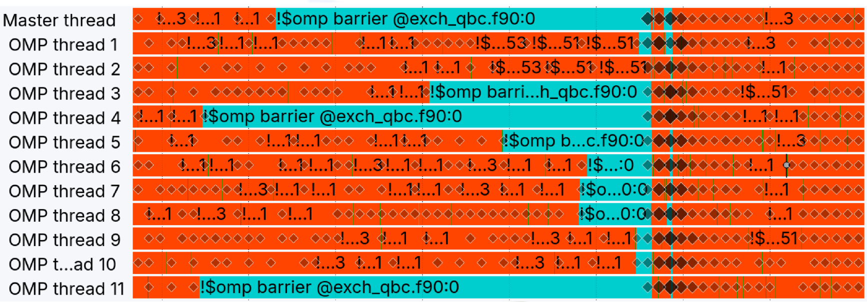


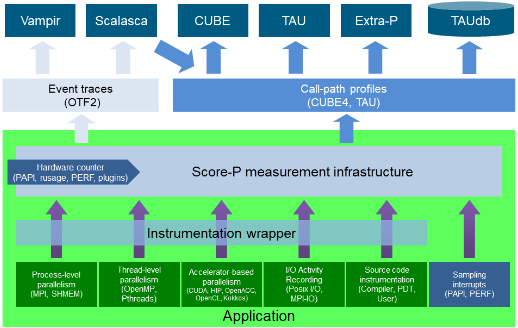
SCORE-P AND OMPT

Smoothing the bumpy road to OpenMP performance measurement

September 19, 2024 | Jan André Reuter | Jülich Supercomputing Centre (JSC)

WHAT IS SCORE-P?

- Score-P is a highly scalable performance measurement tool
- Support for multi-process, thread-parallel and accelerator-based paradigms
- Support for additional metrics (I/O, HW counters, . . .)
- Flexible measurement without re-compilation:
  - Profile generation (CUBE4 format)
  - Event trace recording (OTF2 format)
- Support for C, C++, Fortran and Python



OPENMP INSTRUMENTATION

Two adapters for OpenMP

opari2

- Source-to-source instrumentation tool
- Independent from compiler used
- Instrumentation up to OpenMP 3.1
- Various limitations $\rightarrow$ Code sometimes has to be prepared for OPARI2
  - Especially with C++ and some Fortran features

OpenMP Tools Interface

- Standardised tool interface since OpenMP 5.0
- Enables development of tools based on the OpenMP specification
- Support for the latest and greatest OpenMP features
- Continuously expanded with new versions

OPENMP TOOLS INTERFACE CALLBACKS

- Interface offers callbacks for almost all OpenMP events.
- Some callbacks are grouped by similar events (e.g. sync).
- A tool can freely decide how many callbacks it wants to implement.

Please note

- Runtimes may not (fully) implement some callbacks.
- Runtimes may behave differently for the same callback.

→ Careful investigation of runtimes required

ompt_callback_[NAME]
thread_begin
thread_end
implicit_task
parallel_begin
parallel_end
masked
work
dispatch
sync_region
reduction
task_create
task_schedule
lock_init
lock_destroy
mutex_acquire
mutex_acquired
mutex_released
nest_lock
flush

CALLBACK EXAMPLES

- Callbacks transfer most information to tool
- Tools are able to store additional information
- Identification in user code via code pointer, lookup e.g. via `addr2line`
- But: Which arguments are passed for a certain user code?

```
typedef void (*ompt_callback_parallel_begin_t)
     (
  ompt_data_t*          encountering_task_data,
  const ompt_frame_t*   encountering_task_frame,
  ompt_data_t*          parallel_data,
  unsigned int          requested_parallelism,
  int                   flags,
  const void*           codeptr_ra
);
```

```
typedef void (*ompt_callback_work_t) (
  ompt_work_t             work_type,
  ompt_scope_endpoint_t   endpoint,
  ompt_data_t*            parallel_data,
  ompt_data_t*            task_data,
  uint64_t                count,
  const void*             codeptr_ra
);
```

```
typedef void (*ompt_callback_task_create_t) (
  ompt_data_t*          encountering_task_data,
  const ompt_frame_t*   encountering_task_frame,
  ompt_data_t*          new_task_data,
  int                   flags,
  int                   has_dependences,
  const void*           codeptr_ra
);
```

DUMPING RUNTIME INFO: `OMPT-PRINTF`

- Specification gives guidelines, but offers some freedom for runtime implementers.
- To analyze runtimes, we developed a basic tool dumping passed information.
- Support for OpenMP 5.2 & TR13 via separate feature branches.
- Available on GitHub: https://github.com/FZJ-JSC/ompt-printf

README.md

OpenMP Tools Interface Example Tool: ompt-printf

Description

This tool is a simple example of how to use the OpenMP Tools Interface (OMPT) to collect information about the execution of an OpenMP program.

The OpenMP Tools Interface is part of the OpenMP standard since version 5.0 and allows programs like performance and correctness tools to collect information such as parallel regions, tasking, worksharing, offloading and more. More information can be found in the documentation.

This tool implements the callback-based part of the OMPT interface. The tool is registered through the standardised `ompt_start_tool` function:

```
extern "C" ompt_start_tool_result_t *
ompt_start_tool( unsigned int omp_version,
                 const char* runtime_version )
```

Here, we decide between three different modes of operation, which can be controlled via the environment variable `OMPT_PRINTF_MODE`:

Mode	Description
0	Disable the tool entirely
1	Enable tool, but print no information
2	Print all events, but without arguments
3	Print all events with arguments

These modes are implemented via C++ templates. This should keep the overhead of the tool as low as possible when choosing between the different modes. By default, the tool is built with mode 2.

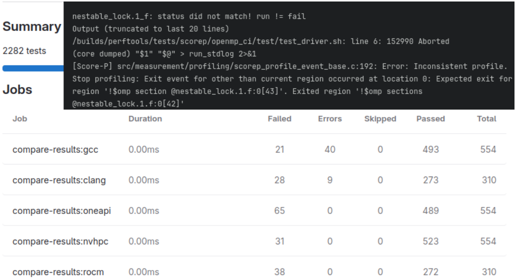
Requirements

These are the requirements to build the tool with the available build system. The library can also be built manually, but is not covered here.

- CMake 3.10 or newer
- A C++17 compliant compiler
- An OpenMP runtime supporting the OMPT interface (e.g. LLVM/Clang, oneAPI, NVHPC, ...)
- CMake will only check the presence of the `omp-tools.h` header file. Actual runtime support is not checked.

CHECKING SCORE-P AND RUNTIMES

- Internal OpenMP CI, built with:
  - Official OpenMP 5.2 examples
  - Additional tests for tasks and teams
  - Regression and smoke tests
- In total 554 tests, with more coming:
  - 310 C / C++
  - 244 Fortran
- Tests uninstrumented and instrumented program runs, includes `ompt-printf`
- Allows for quick comparison of new features and compiler versions



Summary

2282 tests

```
nestable_lock.1_f: status did not match! run != fail
Output (truncated to last 20 lines)
/builds/perftools/tests/scorep/openmp_ci/test/test_driver.sh: line 6: 152990 Aborted
(core dumped) "$1" "$@" > run_stdlog 2>&1
[Score-P] src/measurement/profiling/scorep_profile_event_base.c:192: Error: Inconsistent profile.
Stop profiling: Exit event for other than current region occurred at location 0: Expected exit for
region '!$omp section @nestable_lock.1.f:0[43]'. Exited region '!$omp sections
@nestable_lock.1.f:0[42]'
```

Jobs

Job	Duration	Failed	Errors	Skipped	Passed	Total
compare-results:gcc	0.00ms	21	40	0	493	554
compare-results:clang	0.00ms	28	9	0	273	310
compare-results:oneapi	0.00ms	65	0	0	489	554
compare-results:nvhpc	0.00ms	31	0	0	523	554
compare-results:rocm	0.00ms	38	0	0	272	310

RUNTIME BUGS AND THEIR EFFECT ON SCORE-P

```
[0][Enter: parallel_begin]
    parallel_data = 0 (0x25bfc0)
[0][Exit: parallel_begin]
    parallel_data = 666000001 (0x25bfc0)
[0][parallel_end]
    parallel_data = 666000001 (0x25bfc0)
[0][Enter: parallel_begin]
    parallel_data = 666000001 (0x25bfc0)
[0][Exit: parallel_begin]
    parallel_data = 666000002 (0x25bfc0)
[0][parallel_end]
    parallel_data = 666000002 (0x25bfc0)
```

Minor issues,
e.g. reusing data

```
[0][lock_init] kind = lock
[0][mutex_acquire] kind = lock
[0][mutex_acquired] kind = lock
[1][mutex_acquire] kind = lock
[0][mutex_released] kind = lock
[1][mutex_acquire] kind = lock
[1][mutex_acquired] kind = lock
[1][mutex_released] kind = lock
[0][lock_destroy] kind = lock
```

Remediable issues,
e.g. missing test lock information

```
[0][parallel_begin]
[0][implicit_task] endpoint = begin
[0][work] type = loop | endpoint = begin
[0][work] type = loop | endpoint = end
[1][thread_begin] type = worker
[1][implicit_task] endpoint = begin
[1][work] type = loop | endpoint = begin
[1][work] type = loop | endpoint = end
[1][sync_region] endpoint = begin
[0][sync_region] endpoint = begin
[0][sync_region] endpoint = end
[0][implicit_task] endpoint = end
[0][parallel_end]
[1][sync_region] endpoint = end
[1][implicit_task] endpoint = end
[1][thread_end]
```

Critical issues,
e.g. missing end event

Around 70 OMPT-related bugs were reported since Dec. 2022

JÜLICH Forschungszentrum

RUNTIME SUPPORT

- Widely adopted runtime support by vendors
- Almost all runtimes still have minor issues

```
OMPT support:                   yes
  OMPT header:                  yes
  OMPT tool:                    yes
  OMPT C support:               yes
  OMPT C++ support:             yes
  OMPT Fortran support:         yes
  OMPT critical checks passed: \
                                yes
  OMPT remediable checks passed: \
                                no, wrong_test_lock_mutex, missing_work_loop_schedule detected
  OMPT is default:              yes
```

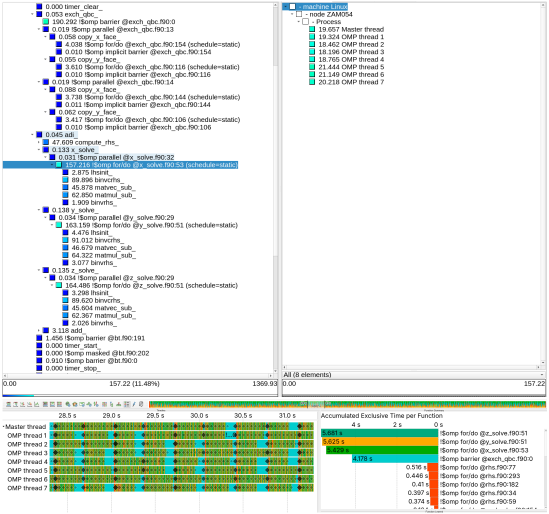
Summary for Intel oneAPI 2024.1.0

Compiler	Host Events
GCC 14.2	None
CCE 17.0.1	Partial
Clang 19.1.0	Full
NVHPC 24.7	Full
oneAPI 2024.2.1	Full
ROCm 6.2	Full

EXAMPLE – BT-MZ

- System:
  - Ubuntu 22.04 LTS
  - Intel Core i7-1260P, 4P+8E cores
  - `flang-new` 19.1.0-rc3
  - Score-P `4d9083fd` (Aug. 20th 2024)

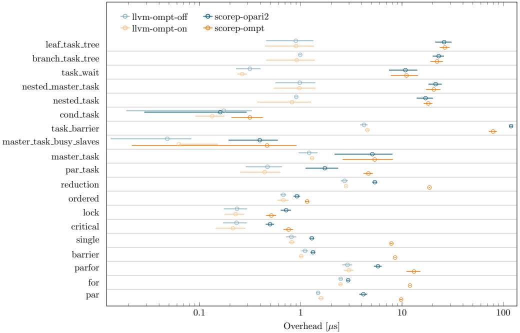
```
BT-MZ Benchmark Completed.
Class           =                        C
Size            =            480x  320x 28
Iterations      =                      200
Time in seconds =                    83.11
Total o_threads =                        8
Total threads   =                        8
Mop/s total     =                 29202.38
Mop/s/thread    =                  3650.30
Operation type  =           floating point
Verification    =               SUCCESSFUL
Version         =                    3.4.3
Compile date    =              26 Aug 2024

Compile options:
   FC           = scorep-flang-new
   FLINK        = $(FC)
   F_LIB        = (none)
   F_INC        = (none)
   FFLAGS       = -O3 -fopenmp
   FLINKFLAGS   = $(FFLAGS)
   RAND         = (none)
```



JÜLICH Forschungszentrum

WHAT ABOUT OVERHEAD?

- Runtime has to call the registered tool callbacks for each event
- A tool handles the events, causing overhead
- Altogether, we want to have a low overhead for accurate measurements
- In 2019, a low overhead was identified without a tool. OMPT caused noticable overhead, but within acceptable range.



Score-P and OMPT, IWOMP 2019

OVERHEAD: TEST SETUP (1/2)

- 1 JURECA-DC CPU node
  (2× AMD EPYC 7742, 512 GiB DDR4)
- Stage 2024 JSC software stack
  - AOCC 4.1.0
  - Clang 16.0.6
  - NVHPC 23.7 (with & without OMPT)
  - oneAPI 2023.2.0
- Benchmarks via JUBE benchmarking environment

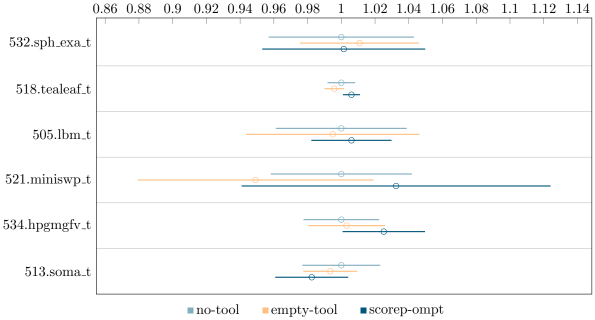
OVERHEAD: TEST SETUP (2/2)

- Selected benchmarks:
  - 6 SPEC HPC 2021 benchmarks
  - 22 EPCC OpenMP 4.0 benchmarks
- Test scenarios:
  - Uninstrumented
  - With `ompt-printf` mode `1`: No output
  - Instrumented with Score-P (`a0b8195e`, July 18th 2024)
- Five runs per toolchain, per benchmark, per test scenario

JÜLICH Forschungszentrum

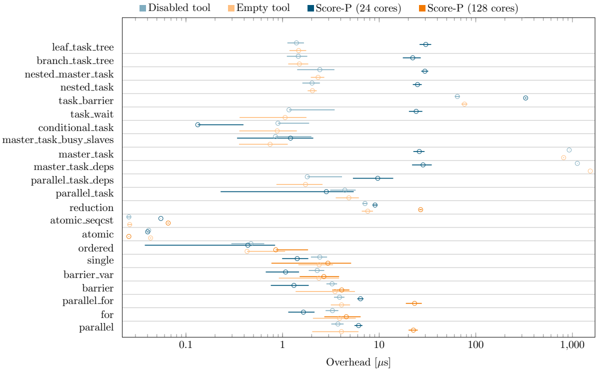
OVERHEAD: SPEC HPC 2021 (AOCC 4.1.0)

- SPEC HPC benchmarks are mostly compute reliant
- Overhead visible, but in acceptable range
- High variation between benchmark results, across all compilers
    - Especially `532.sph_exa_t` and `521.miniswp_t`



OVERHEAD: EPCC (OPENMP 4.0, CLANG 16.0.6)

- Parallel regions cause noticable overhead
  - EPCC has a very low computation amount
  - Score-P needs to handle overdue events
  - Affects task tests as well
  - Dependent on number of threads
- Task tests did not finish for 128 cores
  - Score-P reported out of memory
- Otherwise low overhead for handling directives



DISCUSSION (1/2) – OVERHEAD

- Score-P has a low overhead for computation intensive applications.
- Parallel regions cause high overhead due to overdue event handling.
- Many (very small) parallel regions can skew results.
- Many tasks can abort measurement: Improvements to memory handling required.

DISCUSSION (2/2) – OPENMP TOOLS INTERFACE

- OMPT is a huge leap forward for tool developers compared to OPARI2.
- Analyzing runtime releases for changes and bugs is a major task.
- It took many bug reports and tests to have a working and stable adapter.
- History may repeat itself with OpenMP 6.0
- Our wish: Close collaboration between tool and runtime developers.

Thanks for your attention!
Any questions?

ACKNOWLEDGEMENTS

- This work is funded by the BMBF as part of the ENSIMA project (Grant No. 16ME0630)
- We acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LUMI, hosted by CSC (Finland) and the LUMI consortium through a EuroHPC Regular Access call.