Score-P with Arm(s) around the world ...

Christian Feld
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
Jülich, Nordrhein-Westfalen
Germany
c.feld@fz-juelich.de

Gregor Corbin
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
Jülich, Nordrhein-Westfalen
Germany
g.corbin@fz-juelich.de

Brian J.N. Wylie
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
Jülich, Nordrhein-Westfalen
Germany
b.wylie@fz-juelich.de

Abstract

Now that the first Arm-based exascale computing systems have been deployed, it is more important than ever to tune and scale up HPC applications to fully exploit the available hardware resources. Therefore, there is a strong need for software tools that can assist application developers with this task. The Score-P performance tools ecosystem plays a major role in filling this gap. It consists of the highly scalable Score-P instrumentation and measurement infrastructure for profiling and event tracing of massively parallel HPC application codes, as well as complementary analysis tools that work on the common data formats CUBE4 and OTF2, allowing users to gain insights into their applications' communication, synchronization, input/output, and scaling behavior, pinpointing performance bottlenecks and their causes. Score-P aims to be easy to use, originating from x86-64, SPARC and POWER systems and now also supporting Arm-based systems. In this article, we detail our experiences configuring and installing Score-P with various compiler and MPI combinations on the Fujitsu A64FX systems Fugaku and Deucalion, as well as the Nvidia Grace-Hopper exascale system, JEDI/JUPITER. Moreover, we explore differences in run time and measurement overheads between the systems and toolchains. Finally, we present the results of a performance assessment of the Neko CFD code on JEDI, leveraging the Scalasca, CubeGUI and Vampir analysis tools. We thus demonstrate that Score-P is ready to be used in the current Arm environments, identify remaining limitations and further opportunities to improve the software.

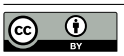
CCS Concepts

• **General and reference** → **Performance**; *Measurement*; • **Computer systems organization** → Reduced instruction set computing; • **Software and its engineering** → *Software performance*.

Keywords

instrumentation, application execution measurement, profiling, event tracing, parallel performance analysis

ACM Reference Format:
Christian Feld, Gregor Corbin, and Brian J.N. Wylie. 2026. Score-P with Arm(s) around the world .... In *SCA/HPCAsia 2026 Workshops: Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region Workshops (SCA/HPCAsiaWS 2026), January 26–29, 2026, Osaka, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3784828.3785348



SCA/HPCAsiaWS 2026, Osaka, Japan

ACM ISBN 979-8-4007-2328-5/26/01
https://doi.org/10.1145/3784828.3785348

1 Introduction

The computational power of Arm-based high-performance computing (HPC) systems is constantly increasing. Jülich Supercomputing Centre's JUPITER is the world's first Arm-based exascale computing system in production use. This immense computing performance comes at the cost of increased hardware complexity. Contemporary Arm-based HPC systems consist of a combination of CPUs, often combined with GPUs, as well as multi-layer memory and network hierarchies. Additionally, the number of available parallel application programming interfaces is growing. These interfaces range from vendor-specific, low-level APIs that expose every single hardware feature, to high-level abstractions that promise portability and convenience. This complexity, in both hardware and software, poses significant challenges for HPC application developers when it comes to tuning and scaling their codes. Thus, there is a strong need for software tools that assist with these tasks. To address this need, industry and academia have developed numerous tools over the past few decades.

One such tool is the instrumentation and measurement infrastructure Score-P [15]. Score-P is a highly scalable [26] open-source tool suite that is intended to be easy to use for both *call path profiling* and *event tracing* of HPC applications. Call path profiling provides an aggregated summary of program execution over time and incurs more or less constant memory costs. Event tracing captures and stores individual execution events in chronological order. Memory costs are proportional to duration but enable in-depth analysis by reconstructing dynamic application behavior. Tools in the Score-P ecosystem [5] (see Figure 1) analyze the data collected during application execution to provide deep insights into the performance behavior of massively parallel applications. CUBE is an open-source tool for analyzing and visualizing call path profiles. Vampir is a commercial tool for interactive event trace visualisation and analysis. The Scalasca Trace Tools analyze event traces to pinpoint communication wait states and their root causes.

In this paper, we share our experiences installing the Score-P instrumentation and measurement framework and the Scalasca Trace Tools on several Arm-based HPC systems using the provided compiler and MPI software stacks.

The remainder of this article is organized as follows. In Section 2 we describe the various Arm systems, including the hardware configuration and software environment. We summarize our experiences building Score-P on these systems with emphasis on shortcomings/quirks of the Fujitsu compilers and the necessary workarounds. In Section 3 we discuss exploratory measurement

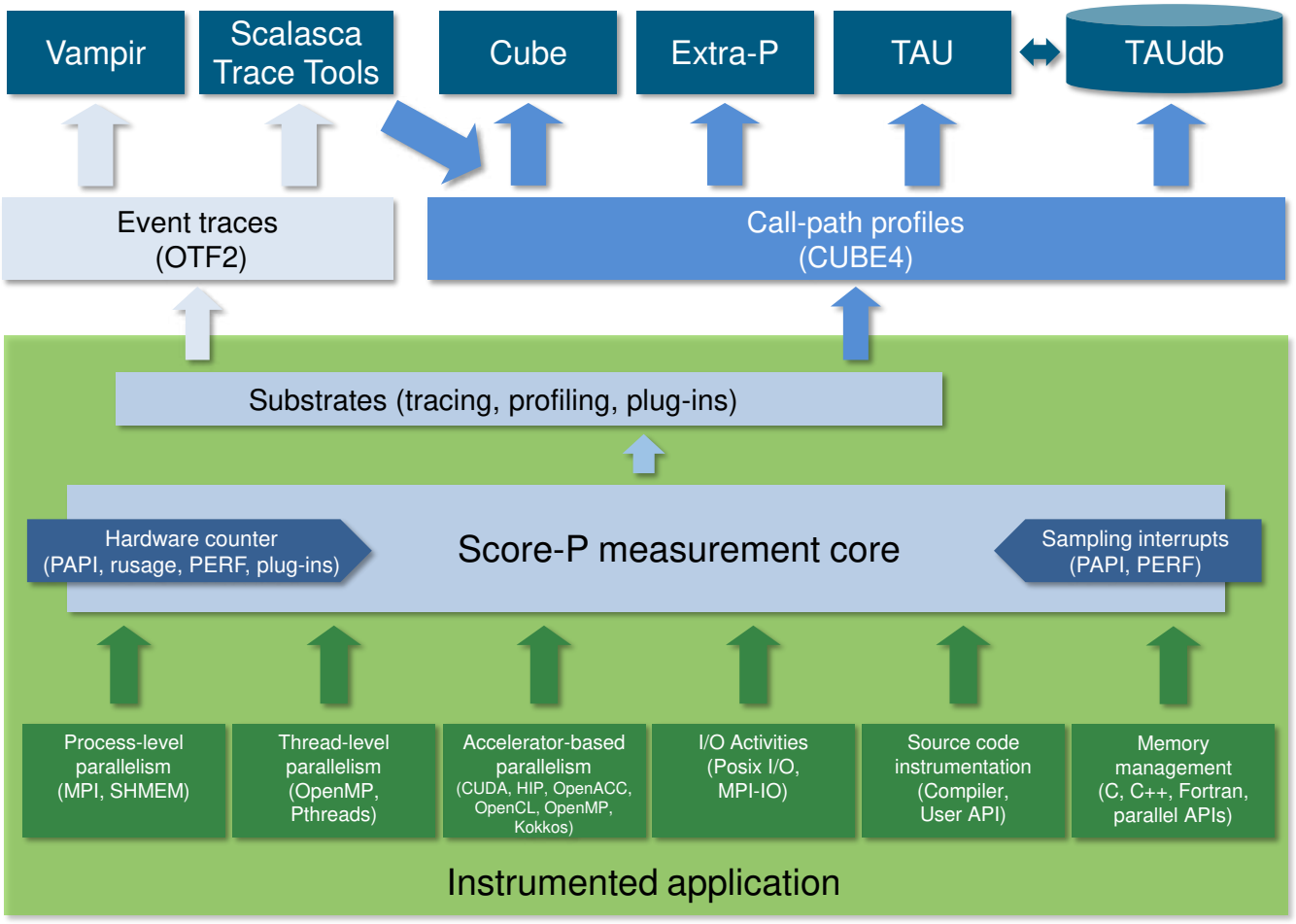


Figure 1: The Score-P performance tools ecosystem overview illustrates the basic data flow between the essential Score-P components, as well as their interaction with various analysis tools through common data formats.

runs done on some of the Arm systems. We compare benchmark run times and measurement overheads on Fugaku and Deucalion, which have identical A64FX hardware but different software environments. We also show performance analysis results for the Neko code on the JEDI test system, which was equipped with Nvidia GH200 chips. Section 4 is dedicated to future work. In Section 5 we give an overview on other tools with Arm support. Finally, Section 6 is for summary and conclusions.

2 Building Score-P

As shown in Table 1, Arm-based HPC systems come in a variety of configurations. The difference between compute node and login node architecture is relevant when building software such as Score-P and Scalasca. The Score-P framework itself consists of a collection of *compute node* libraries and command-line tools. These tools are used to insert instrumentation hooks into applications during compilation and linking, a process typically performed on *login nodes* to avoid wasting precious computing time. The compute node libraries provide the implementation of the instrumentation hooks and the remaining measurement infrastructure. Different CPU architectures for login and compute nodes present a challenge for Score-P's GNU autotools-based [9] build system. Configure-time checks that rely on compile- and link-time checks produce satisfactory results when used with a cross-compiler. This approach was sufficient for the cross-compile machines of the IBM Blue Gene series and the K computer, which both provided restrictive software environments. However, recent compiler features require running and analyzing the results of small test programs to detect runtimes that deviate from the specification [17] and to determine whether to activate the feature. Two prominent examples are OMPT [4]—an API designed by the OpenMP Language Committee to support the construction of portable, efficient, and vendor-neutral performance tools— and the LLVM plug-in [23]—a means for function instrumentation using the LLVM intermediate representation. Test programs cannot be run on a login node if its architecture differs

Table 1: Overview of Arm-based systems on which Score-P has been tested

System	Compute Node	Login Node	Site	Software Environment
JURECA-DC Evaluation Platform	Ampere Altra Q80-30 2 Nvidia A100	x86-64	JSC, Germany	GCC, NVHPC & OpenMPI via EasyBuild
TURPAN	Ampere Altra Q80-30 2 Nvidia A100	Arm	CALMIP, France	ARM, GCC, NVHPC & OpenMPI via `modules`
Irene Arm TDS	Fujitsu A64FX	Arm	CEA, France	Fujitsu, GCC & MPI
Deucalion	Fujitsu A64FX	x86-64	MACC, Portugal	Fujitsu, GCC & MPI via EasyBuild
Fugaku	Fujitsu A64FX	x86-64	RIKEN R-CCS, Japan	Fujitsu, LLVM & MPI via `modules` GCC via Spack
Isambard	2 Cavium ThunderX24	ThunderX24	Bristol University, UK	PrgEnv-gnu & CrayMPI via HPE CPE
JEDI/JUPITER	4 Nvidia GH200	Grace-Hopper	JSC, Germany	GCC, NVHPC & OpenMPI via EasyBuild

from that of the targeted compute node. Thus, on machines such as JURECA-DC, Deucalion, and Fugaku, we must build Score-P on a compute node to be able to query and access all available features.

The compiler suite for building Score-P can be selected via configure options. The systems mentioned in Table 1 provide the following compiler suites: GCC, NVHPC, LLVM/Clang, the Fujitsu compiler suites (in traditional and Clang mode), and the Arm compiler collection. Score-P provides presets for GCC, NVHPC, and LLVM/Clang via the configure option `--with-nocross-compiler-suite`. The configure option `--with-custom-compilers` provides a way to inject compiler and MPI specifications for the others. Using the compiler presets with the `configure`, `make`, and `make install` installation process works seamlessly. The available presets, together with various MPI and accelerator programming environments, and support libraries, are regularly tested on real-world HPC systems [7] as well as customized container setups.[1] Recipes for alternative installation methods via package management systems, such as EasyBuild [13] and Spack [12] are maintained by the Score-P developers for official releases. This allows us to install Score-P for GCC, NVHPC, and LLVM/Clang tool chains that offer *established* names for MPI compiler wrappers, without any problems. For compiler suites not included in the preset list, such as Fujitsu in traditional mode and Clang mode, and Arm, the option `--with-custom-compilers` must be used to select the desired compilers and MPI wrappers. Four configuration files are required to specify the compilers and flags: One is for the login node tools, if applicable, and the other three are for the compute node, with one each for the plain compilers, the MPI, and the SHMEM compiler wrappers. Unused files, often those for the login node and SHMEM specification files, are left empty. The `--with-custom-compilers` option also comes to the rescue if MPI compiler wrappers have uncommon names and autodetection fails. For instance, the MPI compiler wrappers provided by Fugaku's `LLVM/llvmorg-21.1.0` module [19] are only available as `mpiclang`, `mpiclang++`, and `mpiflang`. Score-P's heuristics do not recognize these names, so they must be provided.

2.1 Score-P installations on Arm systems

Due to Score-P's tight dependence on compiler and MPI features, a separate build is needed for each combination. Table 2 shows the successful installations of Score-P v9.3 [21] on Deucalion and Fugaku, and v9.0 on JEDI[2]. We describe these systems in more detail, because they were used to obtain the measurement results described in the next section.

Deucalion, hosted by MACC in Portugal, has two x86-64 partitions (one with Nvidia GPU devices) and one Arm partition. The Arm partition has 1632 nodes equipped with Fujitsu A64FX processors. Software is accessible via modules installed with EasyBuild. EasyBuild manages hierarchical dependencies with so-called *toolchains*, which consist of a compiler suite, an MPI implementation, and additional libraries. Selecting a toolchain provides access to the modules that were built with it. A fairly recent version of the gompi toolchain (2024a), which includes GCC (13.3.0) and OpenMPI (5.0.3), is provided. On top of that, an extensive software stack is installed. Existing EasyBuild recipes for gompi can be used without modification. However, no such toolchain or associated software stack is available for the Fujitsu compiler. That is, the Fujitsu compiler and MPI are provided as a plain module (`FJSVstclanga/1.0.21.02a`). To remain in the EasyBuild framework on Deucalion, we developed an Score-P EasyBuild recipe that uses Fujitsu compilers, based on the GCCcore toolchain. The dependencies required by Score-P were previously installed via EasyBuild, using GCCcore.

Fugaku, hosted by RIKEN R-CCS, is Japan's largest HPC system. It draws its computing power exclusively from A64FX processors. Since the node hardware is identical to that of Deucalion, we can directly compare the software on these systems with respect to intra-node performance and overhead. Similar to Deucalion, Score-P's

[1] See, for example, the Score-P development Docker containers [22].

[2] JEDI was decommissioned soon after Score-P v9.0 was released.

dependencies were installed using an available GCC. The Fujitsu compilers are available on Fugaku via the `lang/tcsds-1.2.41` module. The `LLVM/llvmorg-21.1.0` module provides an LLVM-based toolchain. A GCC with Fujitsu MPI toolchain is available via Spack. Score-P was installed manually using the `configure`, `make`, and `make install` nexus.

JEDI was a small test system hosted at FZJ in Germany to prepare for the installation of JUPITER. It consisted of a single rack equipped with the same hardware as JUPITER, 4 Nvidia GH200 superchips per node. Similar to Deucalion, it provided a hierarchical EasyBuild software stack; recipes for Score-P and its dependencies were already available. Now that its purpose has been fulfilled, its processors have been incorporated in JUPITER.

2.2 Fujitsu Compilers

The Fujitsu compilers `fcc`, `FCC`, and `frt` are available on the two Fujitsu A64FX machines, Fugaku and Deucalion. Both machines come with Fujitsu MPI (i.e., `mpifcc`, `mpiFCC`, and `mpifrt`), which is based on Open MPI. To build Score-P on a compute node with this setup, configuration files for all compute node compilers must be provided to `--with-custom-compilers`. Note that separate configuration files are needed for the Fujitsu compilers in *traditional* and in *Clang* mode (`fcc/FCC -Nclang`). These files can be integrated into the EasyBuild workflow on Deucalion as patches to the Score-P EasyBuild recipe.

Previously, Score-P supported Fujitsu compilers that generated SPARC64 code on the K computer. Since then, Score-P's source code changed significantly. These changes include support for the `mpi_f08` module [2], internal refactorings regarding the link dependencies of the Score-P libraries, our OMPT implementation [6], and more. Prerequisites for these new features are checked during *configure* time, revealing the following obstacles and shortcomings with contemporary Fujitsu compilers:

- The Fujitsu compiler does not support OMPT in either traditional or Clang mode due to missing OMPT headers.
- During OpenMP instrumentation via the source-to-source processor OPARI2, intermediate pre-processed files are generated. For C++, the file extension usually is `.ii`. In traditional mode, `FCC` does not create an object file nor issues an error or a warning when instructed to compile such a file.[3] Subsequent instrumentation steps fail. This could be worked around by choosing the `.i` file extension. In Clang mode, `FCC` accepts the `.ii` extension as documented.
- Although the Score-P compute node libraries are primarily written in the C programming language, the MPI part contains Fortran code in order to support Fortran MPI language bindings. Thus, this part is also linked with the Fortran compiler to a shared or static library. At instrumentation time, this library is linked into a user code, probably using a C or C++ compiler. Mixing languages has caused issues in the past, particularly with Cray compilers. Thus, Score-P tries to determine Fortran intrinsic and runtime libraries that are required to successfully link a Fortran program or shared library at configure time using GNU autoconf's `AC_FC_LIBRARY_LDFLAGS` macro [8]. However, these `FCLIBS` libraries do not always solve linking issues as advertised. Instrumentation for static Score-P installations built with Fujitsu succeeds only when the `FCLIBS` libraries are omitted. In shared Score-P installations, the presence or absence of `FCLIBS` has no effect. Instrumenting MPI codes (in any language) with such an installation fails with linking errors. Thus, shared builds only succeed when Fortran is disabled for the entire Score-P installation.
- Support for the Fortran 2008 language bindings of MPI has to be disabled in Score-P. Although Fujitsu MPI supports these bindings, we cannot determine how to pass choice buffers to these routines. In principle, there are two methods: Either these buffers are passed with an argument descriptor, using the `type(*), dimension(..)` syntax of TS29113, or the compiler provides a non-standard language extension to disable type-kind-rank checks. With Fujitsu, there is a mismatch between compiler and MPI: The compiler does not support any ignore TKR syntax, but the MPI library does not use the descriptor method either.
- In addition to the linking issues arising from mixed-language code, the Fujitsu Fortran compiler exhibited other peculiarities [10] that necessitated adjustments to Score-P's configure checks.

Finally, a static build, slight code and configure check modifications led to a successful Score-P installations on Fugaku and Deucalion.

2.3 Compile times on Fugaku

Table 2 shows the different tool chains available on Fugaku. We observed significantly longer build times when using the LLVM module than with the Fujitsu or GCC compiler suites. LLVM builds took roughly twice as long as other builds. This was independent of the language. It was observed for Score-P, which is primarily written in C, as well as for the BT-MZ benchmark, which is written entirely in Fortran. This benchmark will be used for comparisons in the next section.

3 Performance Assessments

This section utilizes the NAS parallel benchmark suite to compare various tool chains on Deucalion and Fugaku, and assesses the Neko portable framework for high-order spectral-element flow simulations on JEDI.

3.1 Comparison of Fugaku and Deucalion using BT-MZ

To compare Fugaku and Deucalion, which have identical A64FX compute nodes, but different software stacks, we used the BT-MZ benchmark from the NAS parallel benchmark suite [14, 24]. This suite is a set of applications designed to test performance of supercomputers. These benchmarks are ideal for initial evaluations because they are relatively quick and easy to build, and they can be run on a wide range of predefined problem sizes. Furthermore, the BT-MZ benchmark has been part of the Score-P tutorials and

[3] This behavior contrasts with the documentation in Fujitsu's C++ Users guide, see *9.1.1.2. Input Files for the Compile Command* [11].

Table 2: Score-P installations on Deucalion, Fugaku, and JEDI for tool chains provided by the systems. Following abbreviations are used: `ffmpi` for Fujitsu compilers with Fujitsu MPI, `gompi` for GCC with Open MPI, `nvompi` for NVHPC and Open MPI, `gpsmpi` for GCC with ParaStation MPI, and `nvpsmpi` fot NVHPC with ParaStation MPI.

System	Toolchain	link	modules	notes
Deucalion	ffmpi	static	FJSVstclanga/1.0.21.02a	–with-custom-compilers[a]
	ffmpi-clang	static	FJSVstclanga/1.0.21.02a	–with-custom-compilers[a]
	gompi	shared	GCC/13.3.0 OpenMPI/5.0.3	
Fugaku	ffmpi	static	lang/tcsds-1.2.41	–with-custom-compilers[a]
	ffmpi-clang	static	lang/tcsds-1.2.41	–with-custom-compilers[a]
	gfmpi	shared	gcc/12.2.0-gcc-8.5.0-nencizh fujitsu-mpi/head-gcc-12.2.0-m35pkzs	
	llvm	shared	LLVM/llvmorg-21.1.0	–with-custom-compilers[b]
JEDI	gompi	shared	GCC/13.3.0 OpenMPI/5.0.5	
	gpsmpi	shared	GCC/13.3.0 ParaStationMPI/5.10.0-1	
	nvompi	shared	NVHPC/24.9-CUDA-12 OpenMPI/5.0.5	
	nvpsmpi	shared	NVHPC/24.9-CUDA-12 ParaStationMPI/5.10.0-1	

[a] fcc/FCC/frt, mpifcc/mpiFCC/mpifrt
[b] clang/clang++/flang, mpiclang/mpiclang++/mpiflang

trainings [28] for many years and measurements have been taken with very large numbers of MPI processes and OpenMP threads (e.g., on IBM Blue Gene Q JUQUEEN). The code is written in Fortran and uses both MPI and OpenMP for hybrid parallelization. We used the latest version (3.4.3) of the benchmark.

Benchmark Configurations. BT-MZ can be compiled for different problem sizes, ranging from workloads for a workstation, to workloads filling entire compute clusters. Class C, which runs about half a minute on a single A64FX node, is suitable for investigating single-node performance, and scaling to a few nodes. The A64FX processor has 48 compute cores, organized into four L2-cache domains of 12 cores each. Each node was therefore filled with four MPI ranks, each with 12 OpenMP threads. To reduce run-to-run variations and increase performance, the OpenMP threads were pinned to a single core each. The Deucalion documentation [3] recommends optimization options for the available compilers. We followed these recommendations and used `-O2 -ftree-vectorize -march=native -fno-math-errno` for GCC and `-O3 -Kfast` for Fujitsu. On Fugaku, BT-MZ's default `-O3` was used.

Preliminary scaling runs on Deucalion, shown in figure 2, demonstrate that Class C scales well up to eight nodes in this configuration. For the following assessments, we used four nodes.

We ran this benchmark on Deucalion and Fugaku for the respective installed toolchains, as listed in table 2. On Deucalion, 'gopmi', 'ffmpi' and 'ffmpi-clang' are available. Additionally, we compiled two variants of the benchmark for the 'gompi' toolchain: 'gompi/f08' and 'gompi/noplugin'. The first variant, using the normal 'gompi' Score-P installation, is built with the Fortran 2008 MPI bindings enabled. The second variant uses a modified Score-P installation without the GCC plug-in for compiler instrumentation. In the absence of the GCC plug-in, Score-P falls back to using the `__cyg_profile_func_[enter|exit]` API for instrumentation. On Fugaku, the toolchains 'gompi', 'ffmpi', 'ffmpi-clang' and 'llvm' are installed. The 'llvm/f08' variant uses the Score-P built for 'llvm', but the benchmark is compiled with the F08 MPI bindings.

First, the benchmarks were run without instrumentation to obtain reference times. Then, profile measurements with Score-P instrumentation and run-time filtering were taken. Finally, profile measurements with compile-time filtering were taken where supported ('gompi' with the GCC plug-in, and 'llvm' with the LLVM plug-in).

Observations. Figure 3 shows the run times for these experiments and the measurement overheads. On Deucalion, GCC produces the faster reference executables. Measurement overheads with run-time filtering are below the generally targeted 5%. With compile-time filtering, we observed even negative overheads. Using the F90 or F08 MPI bindings does not significantly impact performance. The reference executables produced by Fujitsu are about 30% slower. We observe no difference in reference times between Fujitsu compilers in traditional and Clang mode, because the benchmark is written in Fortran and Clang mode only applies to C/C++. However, because Score-P is written in C, changing from traditional to Clang mode can influence the measurement overhead. In traditional mode, the profiling overhead is 81%, in Clang mode it is 34%; both are too large to be acceptable. The instrumentation method has a major impact on run time overhead. 'gompi', using the GCC plug-in, is much faster than 'ffmpi' and 'gompi/noplugin', which rely on `__cyg_profile_func_[enter|exit]` for compiler instrumentation.

On Fugaku, executables produced with GCC 12.2.0 and Fujitsu MPI [18] are more than 25 times slower than those compiled with LLVM or Fujitsu. This renders GCC unusable on the system;[4] therefore, Fugaku results for 'gompi' are omitted from the figure. LLVM

[4] The reason for this unexpected behavior was not investigated. Note that the GCC installation used for the measurements—as described in [18], is now available at `/vol0004/apps/oss/spack-v0.21/share/spack/setup-env.sh` and is no longer available at `/vol0004/apps/oss/spack/share/spack/setup-env.sh`.

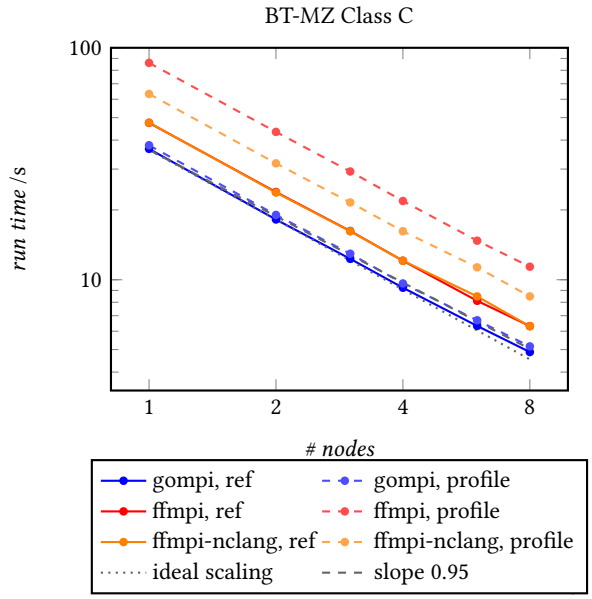


Figure 2: BT-MZ Class C run times on Deucalion for the installed toolchains (gompi, ffmpi, ffmpi-nclang). Solid lines show times for the reference runs (without instrumentation), dashed lines show times for profile runs (instrumented with Score-P, with run-time filtering). The black lines additionally show the theoretically ideal scaling, and a lower bound for the observed scaling (slope 0.95).

produces the program with the fastest reference time, Fujitsu being 36% slower. LLVM on Fugaku is also comparable to GCC on Deucalion, the former being 4% faster. Again, the choice of MPI binding has no noticeable influence on run times. At 11%, the measurement overhead for LLVM, which uses the LLVM plugin, is larger than ideal, but still acceptable. Compile-time filtering can reduce this to 9%. The Fujitsu binary runs 47% slower than the LLVM program. Once again, we observe larger-than-acceptable overheads with Fujitsu, with the Clang mode having the advantage over the traditional mode.

Apart from the defective GCC installation on Fugaku, the open-source tool chains, i.e. 'gompi' on Deucalion and 'llvm' on Fugaku, have the advantage over the Fujitsu-based tool chains. The open-source tool chain versions are fairly recent, while the Fujitsu tool chains simulate the out-of-date compilers GCC 6.1 and LLVM 7.1.0. Score-P's support for GCC and LLVM plug-ins for compiler instrumentation is also a significant advantage over the Fujitsu compiler, resulting in acceptable measurement overheads.

3.2 Assessment of Neko on JEDI

The Neko portable framework for high-order spectral-element flow simulations is the highly-scalable flagship code of the CEEC Centre of Excellence in Exascale Computational Fluid Dynamics. It is modern Fortran parallelized for heterogeneous computers with MPI, OpenMP & CUDA, where each GPU is driven by a dedicated

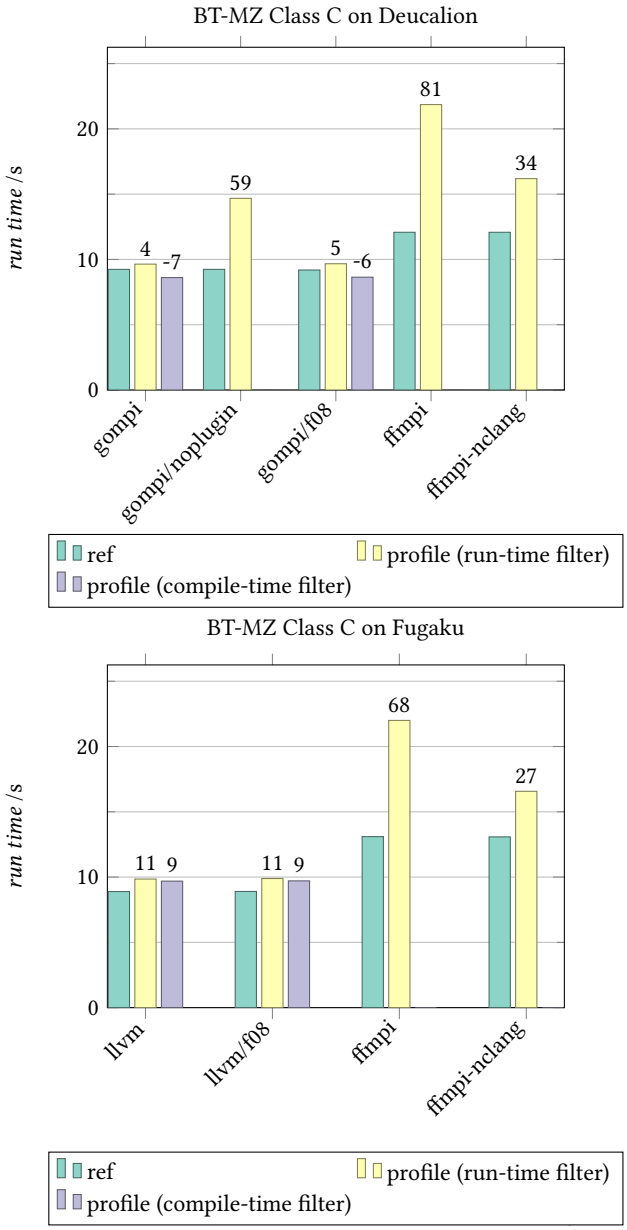


Figure 3: Run times of the BT-MZ Class C benchmark on four nodes: Top: Deucalion; bottom: Fugaku. For each configuration, the time for a reference run and the time for a profile measurement with run-time filtering are shown. If supported, a profile measurement with compile-time filtering is shown. Numbers on top of bars are the overhead in % relative to the respective reference run. Times for 'gompi' on Fugaku are 25 times larger due to a defective GCC installation, and are therefore excluded from the plot.

MPI process. It was built with GCC/13.3.0 and GPU-Aware Open-MPI/5.0.5 with CUDA/12.6.0, instrumented by Score-P/9.0. NVTX annotated regions such as Time-Step within Neko source were automatically captured.

A few initial test measurements were taken, from which a filter specification was produced listing compiler-instrumented routines which could be excluded to reduce measurement overheads (dilation and memory). Applying this specification when re-instrumenting with Score-P resulted in a revised Neko executable with optimized instrumentation of routines by the GCC compiler. An additional measurement filter specification was then used to avoid various unnecessary CUDA API events used within the GPU-Aware MPI (`cuEventQuery`, `cuPointerGetAttributes`, etc.) for a cleaner profile.

Subsequent measurements were made executing the Taylor-Green Vortex (TGV) ExaCB benchmark, comprising 301 solver time-steps of a 3D mesh with 2M elements on up to 32 compute nodes of JEDI [27]. Executions use either a single or a pair of OpenMP threads per MPI process, where the former was determined to generally be faster. Measurement dilation of 7–40% was observed (about twice as high as the run-to-run variability).

A Score-P profile from 32 compute nodes post-processed by Scalasca and presented by the CUBE GUI browser is shown in Figure 4, where the `neko_solve` Time-Step region has been selected from the call-tree in the middle pane (marked in blue); the associated kernels launched from this region are automatically highlit in pink and included in the efficiencies calculated by the POP Advisor[20] in the detached panel below. System locations for MPI processes and OpenMP threads with associated devices and streams is shown in both tree and topology variants. An associated CUBE command-line tool, `cube_pop_metrics`, tabulates the efficiencies of Score-P profiles from strong-scaling executions on 4, 8, 16 & 32 JEDI nodes in Figure 5. Figure 6 shows traces for a single Time-Step region with the Vampir GUI, comparing the single-threaded and the two-threads-per-rank executions of Neko on a single compute node of JEDI.

4 Future Work

Score-P can be made more conveniently accessible on cross-compile systems such as Fugaku and Deucalion by making some, if not all, of Score-P's command line tools available also on the login node. Currently, an interactive session is required to run these tools.

The `__cyg_profile_func_[enter|exit]` compiler instrumentation needs further investigation to understand the overhead and compiler dependency. For the Fujitsu and NVHPC compiler suites, this is currently the only available function entry/exit instrumentation. This method requires a hash-table lookup for every function entry/exit. An alternative method applicable to LLVM-based compilers that support the XRay function call tracing system [1] is under development [16] and promises reduced overhead.[5]

We expect to compare Score-P measurements on Fugaku and JUPITER-Booster (possibly at large scale), which would also include PAPI hardware counter measurements for the Arm CPU cores (with A64FX and GH200 processors).

5 Related Work

Some related work from the latest Tools Guide from the Virtual Institute – High Productivity Supercomputing [25]

- BSC Extrae measurement infrastructure for event tracing and TALP for online analysis of MPI, OpenMP, OpenACC and other parallel programming models using C, C++, Fortran & Python supports Arm CPUs.[6]
- INESC-ID cross-platform cache-aware roofline model tool (CARM) supports Arm AArch64 Neon.[7]
- Linaro (formerly Arm) Forge (DDT, MAP & Performance Reports) supports debugging and profiling C, C++ and Fortran threaded and parallel code on the latest 64-bit Arm CPUs.[8]
- ParaTools TAU Performance System profiling and tracing toolkit for C, C++, UPC, Fortran, Python & Java combined with MPI and multithreading supports Arm CPUs.[9]
- UVSQ modular assembly quality analyzer and optimizer (MAQAO) supports MPI and multithreading on Arm v8+ for Neoverse N1, V1 & V2 processors. Support for AArch64 is currently in beta.[10]
- VŠB/TUOstrava IT4Innovations MERIC energy efficiency suite supports Arm Aarch64 (specifically A64FX).[11]
- Nvidia Nsight Systems is a system-wide performance analysis tool tracing and visualizing Grace-Hopper Arm CPU and GPU interactions using Python, MPI, threading and CUDA.[12]

6 Conclusion

Score-P development endeavors to support Arm processors on a wide variety of HPC systems with different programming environments, despite considerable challenges encountered with some of them. These include Fugaku, which was one of the earliest, through to JEDI and its subsequent incorporation within JUPITER-Booster, the first European exascale supercomputer.

A comparison of Score-P measurement performance with the NPB BT-MZ benchmark on Fugaku and Deucalion is followed by an example of profile and trace measurements of executions on JEDI of the Neko exascale CFD application combining MPI, OpenMP & CUDA examined with Scalasca's CUBE analysis report explorer and Vampir trace analyzer.

Acknowledgments

This work was supported by the *Developer tools for porting and tuning parallel applications on extreme-scale parallel systems* project[13] of the Joint Laboratory for Extreme-Scale Computing (JLESC) and the *Performance Optimisation & Productivity Centre of Excellence*

[5] XRay instrumentation has been successfully tested on x86-64 systems with HPE CCE, IBM, ROCm/AOCC, and Clang, as well as on Arm systems with Clang. However, the required library must be provided. Fujitsu compilers have not yet been tested.

[6] https://tools.bsc.es/extrae
[7] https://github.com/champ-hub/carm-roofline
[8] https://www.linaroforge.com/
[9] https://www.paratools.com/tau
[10] https://maqao.org/
[11] https://code.it4i.cz/energy-efficiency/meric-suite/meric
[12] https://developer.nvidia.com/nsight-systems
[13] https://jlesc.github.io/projects/tool_pandt_project/

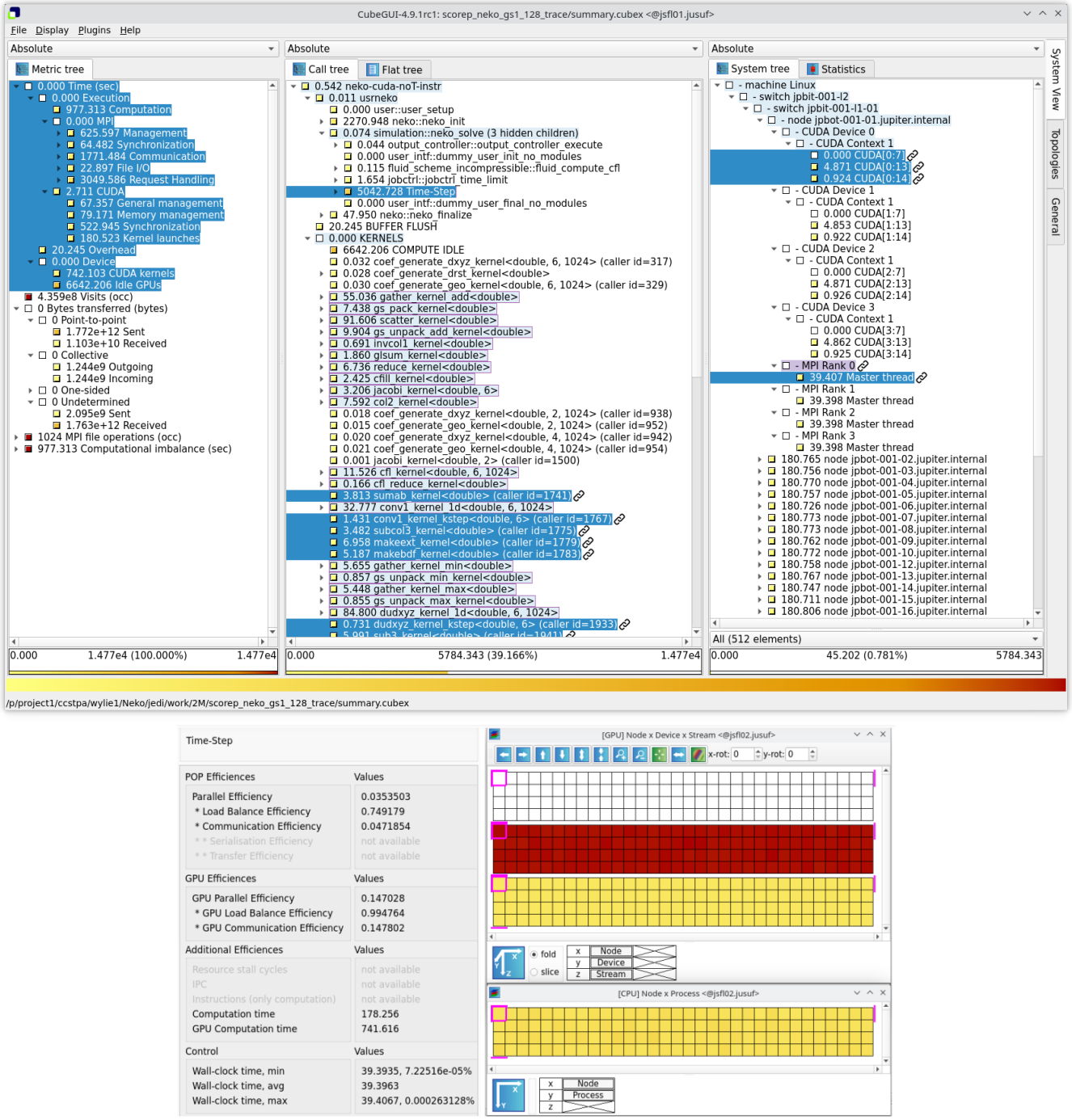


Figure 4: CUBE GUI with post-processed Score-P profile showing system tree with execution time of MPI processes on host CPU cores and associated CUDA streams on GPU devices for selected focus of analysis `neko_solve` Time-Step (and its associated kernels) for an execution of the Neko code with TGV 2M grid benchmark on 32 nodes of JEDI (128 GH200 processors). The MPI process with rank 0 and its associated GPU streams are selected, and these are also highlit in the detached panels below with the topology views of GPU streams and CPU threads for each compute node, showing that these are well load-balanced. The final detached panel shows efficiency metrics calculated for Time-Step, including its CPU and GPU load balance efficiencies (additional greyed out efficiencies are only available from appropriate hardware counter measurements and automated trace analysis by Scalasca).

Compute Nodes CPUs + GPUs	4 16 + 16	8 32 + 32	16 64 + 64	32 128 + 128
POP Metric	**Profile 0**	**Profile 1**	**Profile 2**	**Profile 3**
CPU Parallel Efficiency	0.066019	0.047983	0.046041	0.035350
* CPU Load Balance Efficiency	0.560268	0.794042	0.800327	0.749179
* CPU Orchestration Efficiency	0.117834	0.060428	0.057528	0.047185
* * Serialisation Efficiency	nan	nan	nan	nan
* * Transfer Efficiency	nan	nan	nan	nan
GPU Parallel Efficiency	0.678187	0.545672	0.337415	0.147028
* GPU Load Balance Efficiency	0.998370	0.997429	0.996854	0.994764
* GPU Orchestration Efficiency	0.679294	0.547078	0.338479	0.147802
Additional Metrics				
CPU Resource stall cycles	nan	nan	nan	nan
CPU Instructions/Cycle (IPC)	nan	nan	nan	nan
CPU Instructions (only computation)	nan	nan	nan	nan
CPU Computation time [s]	71.642161	66.498490	100.623483	178.255885
GPU Computation time [s]	736.061048	756.408885	737.557180	741.615604
FOA Quality Control Metrics				
Wall-clock time; min	67.8237, 0.002%	43.3075, 0.004%	34.1469, 0.004%	39.3935, 0.007%
avg	**67.8251**	**43.3092**	**34.1485**	**39.3963**
max	67.8335, 0.012%	43.3187, 0.021%	34.1548, 0.019%	39.4067, 0.026%

Figure 5: Output of the `cube_pop_metrics` command line tool comparing efficiencies of the `Time-Step` region in the Neko TGV 2M benchmark for various allocation sizes. Columns Profile 0, 1, 2, & 3 show the efficiency metrics for runs on JEDI with 4, 8, 16 & 32 nodes, respectively. (Metrics determined from automated trace analysis and CPU & GPU hardware counters are unavailable in this case.)

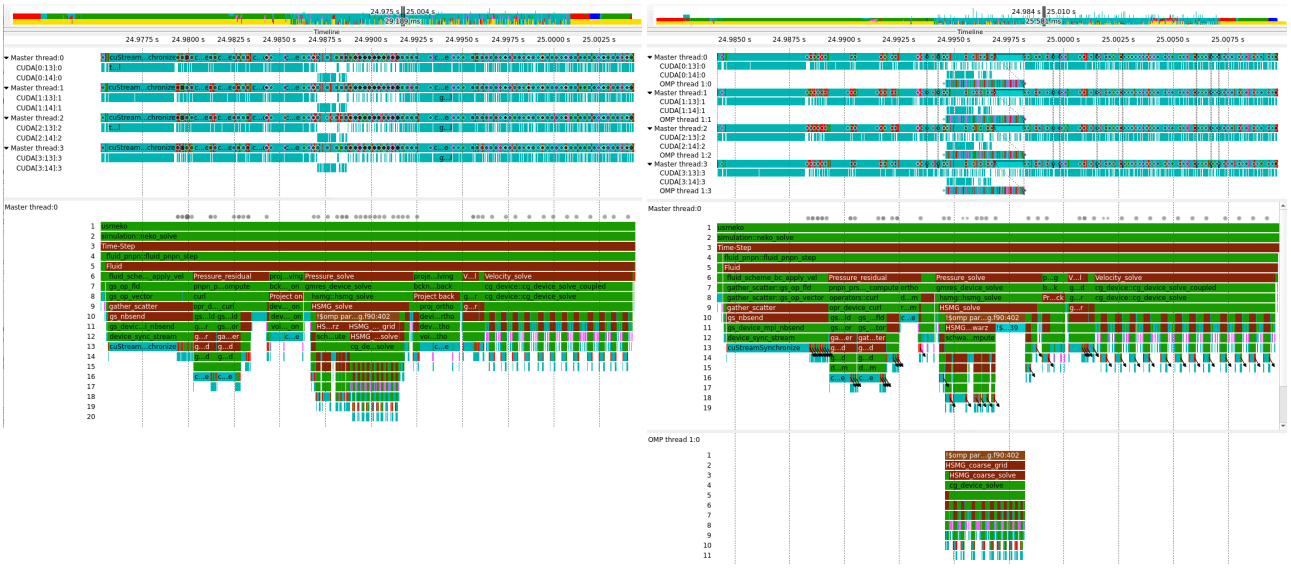


Figure 6: Vampir GUI showing a single timestep of Score-P traces of Neko executions on a single JEDI compute node. Left: Single-threaded execution, right: each of the four MPI processes has two OpenMP threads. Event state timelines for CPUs and GPUs at the top, with callstacks of the thread(s) of MPI rank 0 below. `HMSG_solve` contains the only OpenMP parallel region and this is notably faster with concurrent executions of `HSMG_Schwarz` & `HSMG_coarse_grid`, each with MPI communication and CUDA kernel launches to the same GPU, however, the remainder of the Time-Step is slower when only executed by the primary OpenMP thread.

(POP CoE)[14] for HPC Applications funded by EU Horizon Europe grant 101143931.

Niclas Jansson (KTH) provided the flagship Neko code and execution testcase of CFD for Exascale (CEEC)[15] Centre of Excellence for HPC Applications funded by EU Horizon Europe grant 101093393.

This project received access to JEDI (JUPITER Exascale Development Instrument), a preparation system for the JUPITER supercomputer, through the JUPITER Research & Early Access Programme (JUREAP)[16]. The acquisition and operation of the JUPITER supercomputer is funded jointly by the European High-Performance Computing Joint Undertaking (EuroHPC JU) through the European Union's Digital Europe programme, the German Bundesministerium für Bildung und Forschung (BMBF) and Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen (MKW NRW) via the Gauss Centre for Supercomputing (GCS).

References

[1] Dean Michael Berris, Alistair Veitch, Nevin Heintze, Eric Anderson, and Ning Wang. 2016. *XRay: A Function Call Tracing System.* Technical Report. Google.

[2] Gregor Corbin. 2025. Performance measurements of modern Fortran MPI applications with Score-P. In *Tools for High Performance Computing 2024 - Proceedings of the 15th International Workshop on Parallel Tools for High Performance Computing*, Hartmut Mix, Christoph Niethammer, Bert Wesarg, Wolfgang E. Nagel, and Michael M. Resch (Eds.). Springer International Publishing. (to appear).

[3] Deucalion Support Team. 2025. Deucalion User Guide. https://docs.macc.fccn.pt/compilation/. Accessed October 28, 2025.

[4] Alexandre E. Eichenberger, John Mellor-Crummey, Martin Schulz, Michael Wong, Nawal Copty, Robert Dietrich, Xu Liu, Eugene Loh, and Daniel Lorenz. 2013. OMPT: An OpenMP Tools Application Programming Interface for Performance Analysis. In *OpenMP in the Era of Low Power Devices and Accelerators (LNCS, Vol. 8122)*. 9th International Workshop on OpenMP, Canberra (Australia), 16 Sep 2013 - 18 Sep 2013, Springer, Berlin/Heidelberg, 171–185. doi:10.1007/978-3-642-40698-0_13

[5] Christian Feld, Alexandru Calotoiu, Gregor Corbin, Markus Geimer, Marc-André Hermanns, Maximilian Knespel, Bernd Mohr, Jan André Reuter, Maximilian Sander, Pavel Saviankou, Marc Schlütter, Robert Schöne, Sameer S. Shende, Anke Visser, Bert Wesarg, William R. Williams, Felix Wolf, Brian J. N. Wylie, and Mikhail Zarubin. 2026. The Score-P Performance Tools Ecosystem. *Frontiers in High Performance Computing* (2026). (to appear).

[6] Christian Feld, Simon Convent, Marc-André Hermanns, Joachim Protze, Markus Geimer, and Bernd Mohr. 2019. Score-P and OMPT: Navigating the perils of callback-driven parallel runtime introspection. In *OpenMP: Conquering the Full Hardware Spectrum: 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11–13, 2019, Proceedings 15*. Springer, 21–35.

[7] Christian Feld, Markus Geimer, Marc-André Hermanns, Pavel Saviankou, Anke Visser, and Bernd Mohr. 2021. Detecting Disaster Before It Strikes: On the Challenges of Automated Building and Testing in HPC Environments. In *Tools for High Performance Computing 2018 / 2019 / Mix, Hartmut (Editor)*. 12th International Parallel Tools Workshop, Stuttgart (Germany), 17 Sep 2018 - 18 Sep 2018, Springer International Publishing, 3–26. doi:10.1007/978-3-030-66057-4_1

[8] Free Software Foundation. 2025. Autoconf—Macro: AC_FC_LIBRARY_LDFLAGS. https://www.gnu.org/savannah-checkouts/gnu/autoconf/manual/autoconf-2.72/autoconf.html#index-AC_005fFC_005fLIBRARY_005fLDFLAGS-1. Accessed October 17, 2025.

[9] Free Software Foundation. 2025. An Introduction to the Autotools. https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html. Accessed October 17, 2025.

[10] Fujitsu. 2023. Fujitsu Software Technical Computing Suite V4.0L20—Development Studio Fortran Compiler Messages—J2UL-2562-01ENZ0(07) September 2023. https://www.r-ccs.riken.jp/fugaku/docs/manual/en/lang/j2ul-2562-01enz0.pdf. Accessed October 24, 2025.

[11] Fujitsu. 2025. Fujitsu Software Technical Computing Suite V4.0L20—Development Studio C++ User's Guide—J2UL-2561-01ENZ0(15) March 2025. https://www.r-ccs.riken.jp/fugaku/docs/manual/en/lang/j2ul-2561-01enz0.pdf. Accessed October 20, 2025.

[12] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. 2015. The Spack Package Manager: Bringing Order to HPC Software Chaos *(Supercomputing 2015 (SC'15))*. Austin, Texas, USA. doi:10.1145/2807591.2807623

[13] Kenneth Hoste, Jens Timmerman, Andy Georges, and Stijn De Weirdt. 2012. EasyBuild: Building Software with Ease. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 572–582. doi:10.1109/SC.Companion.2012.81

[14] Jill Dunbar 2025. NAS Parallel Benchmarks. https://www.nas.nasa.gov/software/npb.html. Accessed October 27, 2025.

[15] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. 2012. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Tools for High Performance Computing 2011*, Holger Brunst, Matthias S. Müller, Wolfgang E. Nagel, and Michael M. Resch (Eds.). Springer, Berlin, Heidelberg, 79–91. doi:10.1007/978-3-642-31476-6_7

[16] Sebastian Kreuzer, Paul Adelmann, and Christian Bischof. 2025. A Runtime-Adaptable Instrumentation Back-End for Score-P. In *Tools for High Performance Computing 2024 - Proceedings of the 15th International Workshop on Parallel Tools for High Performance Computing*, Hartmut Mix, Christoph Niethammer, Bert Wesarg, Wolfgang E. Nagel, and Michael M. Resch (Eds.). Springer International Publishing. (to appear).

[17] Jan André Reuter, Christian Feld, and Bernd Mohr. 2025. Score-P and OMPT: Smoothing the bumpy road to OpenMP performance measurement. In *Tools for High Performance Computing 2024 - Proceedings of the 15th International Workshop on Parallel Tools for High Performance Computing*, Hartmut Mix, Christoph Niethammer, Bert Wesarg, Wolfgang E. Nagel, and Michael M. Resch (Eds.). Springer International Publishing. (to appear).

[18] RIKEN Center for Computational Science. 2025. Compilers that generate binaries for compute nodes—Combination of GNU Compiler Collection and Fujitsu MPI. https://www.fugaku.r-ccs.riken.jp/doc_root/en/user_guides/lang_latest/CompileforCN/GCC_MPI/index.html#compiling-on-a-computation-node. Accessed December 15, 2025.

[19] RIKEN Center for Computational Science. 2025. Compilers that generate binaries for compute nodes—LLVM. https://www.fugaku.r-ccs.riken.jp/doc_root/en/user_guides/lang_latest/CompileforCN/LLVM/index.html. Accessed October 20, 2025.

[20] Pavel Saviankou. 2025. Tool Time: Automated POP Metrics with `cube_pop_metrics` and POP_Advisor. https://pop-coe.eu/blog/tool-time-automated-pop-metrics-with-cubepopmetrics-and-popadvisor. Accessed 3 November 2025.

[21] Score-P Development Team. 2025. Scalable performance measurement infrastructure for parallel codes (Score-P), v9.3. https://doi.org/10.5281/zenodo.17297069 [Accessed October 31, 2025].

[22] Score-P Development Team. 2025. Score-P on Docker Hub. https://hub.docker.com/u/scorep. Accessed October 20, 2025.

[23] Ronny Tschüter, Johannes Ziegenbalg, Bert Wesarg, Matthias Weber, Christian Herold, Sebastian Döbel, and Ronny Brendel. 2017. An LLVM Instrumentation Plug-in for Score-P. In *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC* (Denver, CO, USA) *(LLVM-HPC'17)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. doi:10.1145/3148173.3148187

[24] Rob F. van der Wijngaart and Jin Haopiang. 2003. NAS Parallel Benchmarks, Multi-Zone versions. In *Supercomputing 2003*.

[25] Virtual Institute – High Productivity Supercomputing. 2025. VI-HPS Tools Guide. https://www.vi-hps.org/cms/upload/material/general/ToolsGuide.pdf. Accessed 3 November 2025.

[26] Brian Wylie. 2018. *Nekbone MPI+OMP on JUQUEEN performance audit (POP_AR_112)*. doi:10.5281/zenodo.17142583

[27] Brian Wylie. 2025. *Neko MPI+OpenMP+CUDA on JEDI performance assessment (POP3_AR_047)*.

[28] Brian J. N. Wylie, Judit Giménez, Christian Feld, Markus Geimer, Germán Llort, Sandra Mendez, Estanislao Mercadal, Anke Visser, and Marta García-Gasulla. 2025. 15+ years of joint parallel application performance analysis/tools training with Scalasca/Score-P and Paraver/Extrae toolsets. *Future Generation Computer Systems* 162, C (2025). doi:10.1016/j.future.2024.07.050

[14] https://www.pop-coe.eu/

[15] https://www.ceec-coe.eu/

[16] https://www.fz-juelich.de/en/jsc/jupiter/jureap