2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC) | 978-1-6654-6021-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/P3HPC56579.2022.00017

ECP SOLLVE: Validation and Verification Testsuite Status Update and Compiler Insight for OpenMP

1st Thomas Huber
University of Delaware
thuber@udel.edu

2nd Swaroop Pophale
Oak Ridge National Laboratory
pophaless@ornl.gov

3rd Nolan Baker
University of Delaware
nolanb@udel.edu

4th Michael Carr
University of Delaware
mjcarr@udel.edu

5th Nikhil Rao
University of Delaware
nikhilr@udel.edu

6th Jaydon Reap
University of Delaware
jreap@udel.edu

7th Kristina Holsapple
University of Delaware
kris@udel.edu

8th Joshua Hoke Davis
University of Maryland
jhdavis@umd.edu

9th Tobias Burnus
Siemens
Tobias_Burnus@mentor.com

10th Seyong Lee
Oak Ridge National Laboratory
lees2@ornl.gov

11th David E. Bernholdt
Oak Ridge National Laboratory
bernholdtde@ornl.gov

12th Sunita Chandrasekaran
University of Delaware
schandra@udel.edu

***Abstract*—The OpenMP language continues to evolve with every new specification release, as does the need to validate and verify the new features that have been implemented by the different vendors. With the release of OpenMP 5.0 and OpenMP 5.1, new target offload and host-based features have been introduced to the programming model. While OpenMP continues to grow in maturity, there is an observable growth in the number of compiler and hardware vendors that support OpenMP. In this manuscript, the main focus is on evaluating the conformity and OpenMP implementation progress of various compiler vendors such as Cray, IBM, GNU, Clang/LLVM, NVIDIA, and Intel. More specifically, the 4.5, 5.0, and 5.1 versions of the OpenMP specification are analyzed. For our experimental setup, the Crusher and Summit computing systems hosted by Oak Ridge National Lab's Computing Facilities are utilized. The effort of vendor agnostic analysis of these implementations is especially valuable for application developers who are using new OpenMP features to accelerate their scientific codes. Insights are presented into the current implementation status of various vendors, the progression of specific compiler's support for OpenMP overtime, the subset of OpenMP 4.5, 5.0, and 5.1 that is supported by all compilers, and examples of how our test suite has influenced discussion regarding the correct interpretation of the OpenMP specification. By evaluating OpenMP conformity of pre-Exascale computing systems, the aim is to detail progress and status of AMD + Cray ecosystem before the system and their OpenMP implementation is used for mission critical applications when the first Exascale Computer Frontier is made available to applications.**

***Index Terms*—OpenMP, GPU, Offloading, LLVM**

I. Introduction

Seven out of the ten fastest supercomputers in the world are heterogeneous systems [23]. Heterogeneous systems may be comprised of a CPU and an accelerator such as GPUs, FPGAs, APUs, etc., however, the top performing supercomputers tend to opt towards a configuration of CPU and GPU. For two years in a row (June 2020 - June 2022), the Fugaku A64FX supercomputer produced by Fujitsu and ARM and hosted by RIKEN Center for Computational Science held the title for the fastest supercomputer [24] and proved that a CPU only configuration was able to transcend the performance of heterogeneous systems like Oak Ridge National Laboratory (ORNL)'s Summit (IBM Power9 CPU + NVIDIA V100 GPU).

Following the release of ORNL's Frontier, the world's first Exascale supercomputer (HPL score of 1.102 Exaflop/s using 8,730,112 cores) [4], again the top supercomputer in the world is composed of a heterogeneous mix of compute power (3rd Gen AMD EPYC 64C CPUs and AMD Instinct MI250X GPU accelerators). As hardware vendors with heterogeneous systems in the TOP500, HPE Cray, IBM, Intel, NVIDIA, and AMD provide software support for various parallel programming models that allow users to port their parallel applications to accelerators.

Considering the various changes in hardware architecture offerings over the years, programming models and base languages are incorporating parallelism that can effectively use the CPU as well as the GPUs. For a long time CUDA [17] has been the first choice for GPU programming. Developed by NVIDIA, CUDA provides an API to program GPUs that can be used in applications written in C/C++ or Fortran. HIP [2]is AMD's proprietary GPU programming environment. Although CUDA and HIP offer great performance for parallel applications, they often require programmers to rewrite their programs entirely and are platform specific.

2831-3909/22/$31.00 ©2022 IEEE
DOI 10.1109/P3HPC56579.2022.00017

As more vendors are entering the GPU market, portable parallel programming methods are required so that application programmers can run codes on diverse heterogeneous systems, such as Summit and Frontier, without massive re-engineering. Directive-based parallel programming models OpenMP [20] and OpenACC offer an approach that allows users to annotate their serial code in a more straightforward manner and produce parallel versions of their applications that will run on many different architectures.

In preparation for the release of ORNL's Frontier and other US Department of Energy (DOE) funded systems, the DOE Exascale Computing Project (ECP) sought to prepare an Exascale software stack to ensure that mission-critical applications are able to embrace the potential performance boosts offered by newer generations of hardware. OpenMP, a parallel programming library, is one component of this software stack. More features that are valuable to developers continue to be added to the OpenMP specification. The objective of ECP's Scaling OpenMP via LLVM for Exascale Performance and Portability (SOLLVE) team is to ensure OpenMP is compatible with the unique software and hardware requirements of exascale computing and to enable seamless migration of applications to the novel computing system.

As of May 2022, the compilers that offer support for OpenMP offloading features (specification versions 4.0 and later) are AMD, Flang, GNU, HPE, IBM, Intel, LLVM, NVIDIA, and Siemens. While this list of compilers that support offloading with OpenMP is significant, there are far less compilers that have continued to expand their OpenMP implementations for versions 4.5, 5.0, and 5.1. According to the OpenMP website [19], the only compilers that have any coverage of OpenMP 5.0 offloading features are AMD, GNU, HPE, Intel, LLVM, Siemens, and NVIDIA.

The need to validate and verify the compiler implementations of OpenMP features becomes increasingly important now than ever as the Frontier exascale systems are being made available for use to the application and software developers. This paper elaborates on the SOLLVE validation and verification testsuite creation strategy, its workflow, statistics on OpenMP features' coverage and challenges faced while writing tests for corner cases. Results and discussions entail evaluation of compilers' current status of stability and maturity of implementations, types of errors, and discussions that have led to the language committee revisiting the verbiage used in the specification.

II. Background and Motivation

A. OpenMP

OpenMP Specification provides an Application Program Interface (API) to allow programmers to develop threaded parallel codes on shared memory systems. The OpenMP directives or `pragmas` are understood by OpenMP aware compilers while other compilers lacking OpenMP support are free to ignore them. Usually a flag such as `-fopenmp` is required at compile time to activate OpenMP recognition and processing by the compiler. Along with compiler directives, OpenMP also provides library routines and environment variables for explicit control. The OpenMP `parallel` directive generates parallel threaded code where the original thread becomes thread "0". The new league of threads share resources of the original thread and the specific data-sharing attributes of variables can be specified based on usage patterns of the application. A basic usage example of the `parallel` directive is provided in the code-snippet below.

```
int A[N][N], B[N][N], C[N][N];
// initialize arrays
#pragma omp parallel for
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      C[i][j] = A[i][j] + B[i][j];
    }
  }
```

Listing 1: Simple C program using OpenMP for matrix-matrix addition

B. Offloading to Devices

OpenMP device directives such as `target` provide mechanisms for an OpenMP program to offload parallel code and data to *target devices*.

NVIDIA	AMD	CCE Fortran OpenMP	CCE C/C++ OpenMP	LLVM (Clang) OpenMP
Threadblock	Work group	omp teams	omp teams	omp teams
Warp	Wavefront	omp simd	omp parallel	omp parallel
Thread	Work item	omp simd	omp parallel	omp parallel

TABLE I: OpenMP Construct Mapping to GPU. [1]

OpenMP offers three levels of parallelism (*teams*, *threads*, and *SIMD lanes*), but existing devices may provide different levels of parallelism (typically two or three levels), and different OpenMP implementations can choose different parallelism mapping for the same target device. Table I lists the equivalence in terminology across different vendors and OpenMP. For example, many of the existing OpenMP compilers largely ignore SIMD clauses when targeting GPUs (mapping OpenMP teams to GPU thread blocks and OpenMP threads to GPU threads), but typical GPUs also expose thread-scheduling units, such as *warps* in NVIDIA GPUs and *wavefronts* in AMD GPUs, and thus other OpenMP compilers may choose to provide fine-grained three level parallelisms (e.g., OpenMP teams to GPU thread blocks, OpenMP threads to GPU warps, and OpenMP SIMD lanes to GPU threads). When targeting CPUs, however, the SIMD clause may play an important role, and most existing OpenMP compilers exploit the SIMD-level parallelism by mapping OpenMP threads to CPU threads and OpenMP SIMD lanes to CPU SIMD lanes. But the applicability of the SIMD parallelism largely depends on the compiler's vectorization capability, and it is still implementation-defined how to map the three level OpenMP parallelisms to CPU parallelisms. Therefore, this work will primarily focus on the functional portability of the existing OpenMP implementations.

OpenMP provides a relaxed-consistency, shared-memory model for a given device, which allows all OpenMP threads to

access the device memory to store and retrieve variables. In the OpenMP device data environments, each device has its own device data environment, which may or may not share storage with other devices. OpenMP device directives offer various data-mapping options (via `map`) to specify how an original variable is mapped from the current task's data environment to a corresponding variable in the target device data environment.

C. New 5.X Features

As newer architectures continue to evolve, so does the feature requirements of parallel applications. To accommodate these needs, the OpenMP ARB continues to add new features to the specification. One of the more intriguing features that was added to the 5.0 specification is `metadirective`, which allows a program to run different variants of an OpenMP directive as determined by a conditional statement. The metadirective provides the `when` clause, which receives arguments like `arch` (architecture) and `isa` (instruction set architecture). A common use case for this directive would be when the architecture is NVIDIA or `when(arch==nvidia)` we can call an OpenMP directive, say `#pragma omp target`. When this condition is not met, we can instead define a default behavior such as `#pragma omp parallel`. In 5.1, the `error` directive and `nothing` directive were added specifically for usage with the metadirective clause, and enable run time errors or non-action behaviors to occur when a condition in the `when` clause is not met.

OpenMP 5.0 was released in November 2018 and it introduced a wide variety of improvements for heterogeneous target offload and host based features. A new addition, the `requires` directive, allows the programmer to request features from the implementation that must be supported to enable proper execution of kernels in a given computation unit. Of these features available for enforcement, reverse offload and unified shared memory prove to the most valuable as they enable on-host execution initiated from the offload device and utilization of a shared memory space between devices, respectively. Another important feature released in OpenMP 5.0 is the `declare mapper` directive. The `declare mapper` directive now allows the creation of user-defined mappers to avoid ambiguities that can arise between explicit and implicit mapping of variables as well as the ability to map members of a struct or class.

The `declare target` directive, which was initially introduced in 4.0, allows the user to explicitly ensure that procedures and global variables can be accessed on a device. The 5.0 specification extends this directive with additional functionalities and clauses. For example, the new clause, `device_type` allows a user to create only device version, only host version, or both versions of a function that they wish to be included in the device memory. Functionality induced by this clause is somewhat similar to `metadirective` in that a user can make a host only or device only version of a function or global variable accessible. However, `metadirective` offers the added bonus of triggering different behavior based on a conditional statement.

Many of the features introduced in 5.0, such as `metadirective` and `requires`, are implementation-dependent, meaning compiler vendors have some variability in the manner by which they choose to implement these features. The `requires` directive inherently requests that an implementation must be able to provide a certain behavior in order to compile and run a program correctly. The certain behaviors that can be requested or 'required' by the programmer are reverse offloading, unified address, unified shared memory, atomic default memory ordering, and dynamic allocators. A user can request reverse offloading using `#pragma omp requires reverse_offload` at the top of their program. If the implementation does not have support for this feature, the program will either ignore the `requires` statement and issue a compiler warning or rather issue a compile error.

The `declare variant` directive, again, can be utilized to achieve a similar functionality as the `metadirective` and `declare target` directive. Utilizing the same context-selector-specification field as the `metadirective`, `declare variant` can call a different version of a provided base function, based on the context or conditional statement with which the directive is associated.

OpenMP 5.1, released in November 2020, introduces features such as the `assume`, `nothing`, `scope`, `interop` directives, loop transformation constructs, new modifier clauses that extend the `taskloop` construct, newer support for indirect calls to the device version of a function in target regions, amongst others.

OpenMP 5.2 was released in November 2021 and continues to add onto the previous OpenMP developments. OpenMP 5.2 specifically made improvements in its memory allocators, use of Fortran PURE procedures, and use of the `scope` construct. OpenMP 5.2 also includes simplified unstructured data offload use, extended support of user-defined mappers, more consistent `linear` clause, and refined OpenMP directives syntax.

The main objectives of this paper can be listed as follows:

1) Describe the SOLLVE V&V Suite, it's intent and scope
2) Provide statistics on OpenMP features' coverage by different vendors
3) Deiscuss the subset of OpenMP which is supported by all compilers referenced in this study
4) Discuss challenges faced while writing feature tests for OpenMP
5) Discuss impact of the SOLLVE V&V testsuite on the user community, OpenMP language specification and applications

III. SOLLVE Validation and Verification Suite

The SOLLVE Validation and Verification testsuite was built to provide open-source vendor agnostic feature tests for the latest OpenMP Specifications with focus on features of interest to applications. The process for collecting application input

across ORNL and other DOE labs is outside the scope of this paper. Like most software projects, SOLLVE V&V is aware that with vendor implementations and application changes, the importance of features may change over time and we perform regular checks and add missing tests/corner cases to the testsuite.

A. Test Creation Strategy

Within every new release of the OpenMP specification, there is a section that details the differences between the most current version and its predecessor, which outlines all of the new features that are provided to users. Compiler developers from LLVM, GNU, and more take this differences list, or a similar list potentially provided by the OpenMP ARB, and formulate a to-be-implemented list that is typically hosted on their website. For LLVM, each of the features will have a status associated with it that describes the progress made thus far in implementing: either unclaimed, worked on, mostly done, not upstream, or done. Our development of feature tests is dictated first by the ECP application needs. Through our interactions with the Application Development (AD) teams, a priority list of the most desired new features was created. When writing a new test for an already implemented feature, the usability of the feature that is outlined by the specification is analyzed. Then, the potential combinations of options that may be presented are outlined. For example the `default` clause which has options for `shared`, `none`, as well as `firstprivate` & `private` which were introduced in OpenMP 5.1. For this example, there would need to be two tests to encapsulate the full functionality of this new feature. Lastly, careful attention is paid to the 'restrictions' section of each feature, to ensure that the test being written does not violate boundaries that have been outlined by the OpenMP Specification. In the case of OpenMP features that do not have implementations, generating a brand new test that is both syntactically correct and accurate can prove difficult.

In either case it is ensured that the test meets all conditions and restrictions noted in the specification and provides adequate error checking in case of failure. During this stage of development it is common to receive feedback from other collaborators on how to approach or improve the test, especially on new tests that do not have implementations. After the test has been written it enters a review process. This process includes having each test independently verified by two other collaborators, internal or external to our team. Other interested parties, including members form the OpenMP community or ARB, are welcome to provide their input in the form of Github issues or pull-requests.

B. OpenMP 5.x+ Feature Coverage

As of this writing the V&V testsuite includes 258 5.0 tests, 53 5.1 tests & 6 5.2 tests. Since the primary focus of the SOLLVE project is development of OpenMP support in Clang, the testsuite has more coverage in C/C++. The 5.0 coverage includes a mix of Fortran & C versions while the majority of our 5.1 tests are coded in C. Tests have been written for a vast majority of 5.1 features. Test priority related to 5.1 features are based primarily based on the needs of SOLLVE application developers, starting with high-priority, implemented features, and working our way to low-priority non-implemented features. For the new features introduced in OpenMP Specification a) 5.0 SOLLVE V&V testsuite has 100% coverage for C/C++, 70% coverage for Fortran, b) 5.1 Specification 85% coverage for C/C++, 5% coverage for Fortran, and c) 5.2 Specification we have 20% overall coverage. As mentioned before, the objective is not to have feature tests for all the new features but to have tests that cover, in sufficient detail, the important features as indicated by the ECPDOE applications.

C. Challenges

1) Testing Unimplemented Features

Often times tests cases must be written for new OpenMP features that are not yet implemented by any of the major compiler vendors. This makes the OpenMP specification one of the only resources available to understand how the feature would work once implemented. This can lead to some issues when attempting to develop strong tests for new features. A prime example of this is the test case for the `nothing` clause extension of the `metadirective` construct.

Here is an example of a simple implementation:

```
#pragma omp metadirective \
     when( device={arch("nvptx")}: nothing) \
     default( parallel for )
     {
        for (int i = 0; i < N; i++) {
           A[i] += 2;
        }
     }
```

Listing 2: Simple usage of the nothing clause with the metadirective

At a high level, the metadirective provides a way to dynamically change what OpenMP constructs are rendered. In the example above, if the code is offloaded to an NVIDIA device then the `nothing` clause would be rendered. If not, it would default to a `parallel for` loop. At first glance this seems pretty intuitive. Our initial interpretation of the specification was that the `nothing` clause would simply negate the code. In other words, it would not run the for loop if the code was running on an NVIDIA device. Based off the specification itself and various examples this seemed to be correct. This then lead to the bigger question of how to properly test nothing.

This test was initially written by looking for any spawned threads, or signs that the array had been changed and the code had run despite the `nothing` clause. This seemed promising, but it was discovered the team's interpretation was not the same as the compiler implementation, as the specification was vague. The `nothing` clause when used outside of the metadirective implies that OpenMP would ignore the `pragma` statement. However, the code would still run in serial. This meant the initial version of the test was wrong and needed to be amended.

Ultimately the test was reworked to determine if the metadirective had properly used the nothing directive by checking if the code was running in parallel instead of just checking for threads by leveraging the runtime `omp_in_parallel` function. This function would only return 1 if the code is running in parallel. In the example above, that would mean it would only be 1 if the nothing directive was not rendered properly. This ended up being a much more robust way to test the `nothing` clause with metadirectives and was utilized in the final version.

The `nothing` metadirective test demonstrates the challenge of the SOLLVE team's interpretation of a test compared to a compiler vendor's interpretation, and illustrates the need for heavy review and rewriting of code.

2) Unclear Specification

Another example of confusion relating to interpretation of the specification arose when writing a test case for the `has_device_addr` clause, added to the `target` construct in OpenMP 5.1. The description of this clause states: "The `has_device_addr` clause indicates that its list items already have device addresses and therefore they may be directly accessed from a target device." [18] While this may seem straight-forward, the purpose of this is relatively unclear. Questions arise, such as whether these list items be mapped first, and then marked as on the device? If the list items are already on the device, what is the benefit of listing them under the clause? How is it ensured that the list items are not unmapped at the end of a device region so that they remain when utilizing the clause? The difference between `use_device_addr` and `has_device_addr` is not clearly stated in the specification.

Furthermore, this clause was not listed on the OpenMP examples document. [5] This document is often used by the SOLLVE team to assist in creation of tests that have no yet been implemented, as that document is the only official resource supported by the OpenMP ARB which shows the intended purpose and proper syntax of a new feature.

```
#pragma omp target enter data map(to: x, arr)
  #pragma omp target data use_device_addr(x, arr)
  #pragma omp target map(from:
    second_scalar_device_addr,
    second_arr_device_addr) has_device_addr(x,
    arr)
  {
    second_scalar_device_addr = &x;
    second_arr_device_addr = &arr[0];
  }
#pragma omp target exit data map(release: x, arr)
```

Listing 3: Example of has_device_addr directive

The agreed-upon solution for this test arose only after having community-driven detailed discussion on the directive's purpose and the implicit mapping of target directive. It was decided that a `target enter data map` should be used to ensure variables are properly mapped to the device. Then, the `use_device_addr` and `has_device_addr` can be used in tandem to ensure the variables maintain their device addresses in the `target` region.

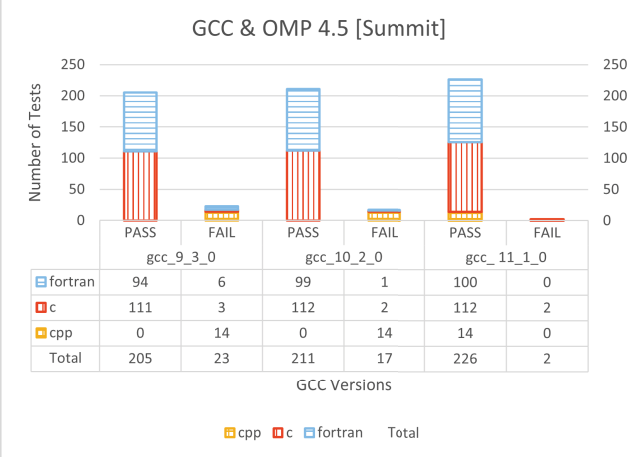


Fig. 1: OpenMP 4.5 tests with GCC on Summit.

IV. Results and Discussion

A. Results from Summit

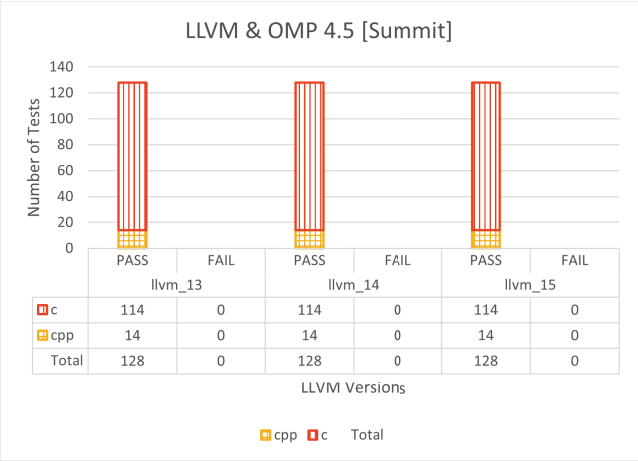
The following subsections shows results of GNU, LLVM and NVHPC compilers and their maturity over time, on Summit.

1) GNU Maturity Over Time

For the GNU results shown in Figures 1, 4 & 7, we only utilize stable releases of the compiler that are made available on OLCF's Summit supercomputer. Regarding the GNU compilers, gcc and g++, there are seemingly a linear increase in support for both 4.5 and 5.0 features in OpenMP across major version releases. It is also important to note that GNU-11.2.0 is the first version of the compiler that supports features described in the OpenMP 5.1 specification. Version 12 Release of GNU supports far more 5.1 features than version 11.2.0, so any users attempting to utilize OpenMP 5.1 and 5.2 features with the GNU compiler should aim to use GNU version 12. Results in Table II show a list of tests that have passed and failed over a set of GCC compiler versions they have been tested on. For OMP 5.0 tests for `loop reduction and/or device` passes on GCC version 9.3.0 but fails in the next two versions i.e. 10.2.0 and 11.1.0.

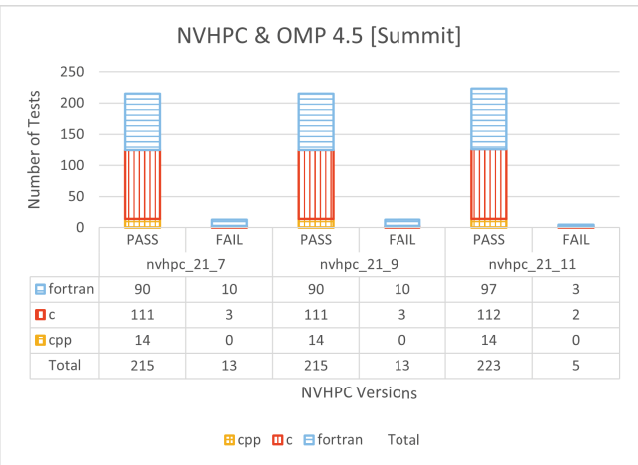
2) LLVM Maturity Over Time

The results presented in Figures 2, 5 & 8, are for Clang and Clang++ using the stable releases of LLVM-13, LLVM-14 and the LLVM-15 developmental release available on the OLCF's Summit supercomputer. It is important to note that LLVM provides an `-fopenmp-version` flag that allows you to inform the compiler which specification version of OpenMP you would like to compile for. This is vital for testing various implementations of features, as many times features will get redefined by new versions of the specification. For example, the `master` construct in OpenMP 5.0 was renamed to `masked` in OpenMP 5.1. The important thing to note here is that LLVM continues to introduce more and more OpenMP 5.1 features. There are some rollbacks in 4.5 and 5.0 which



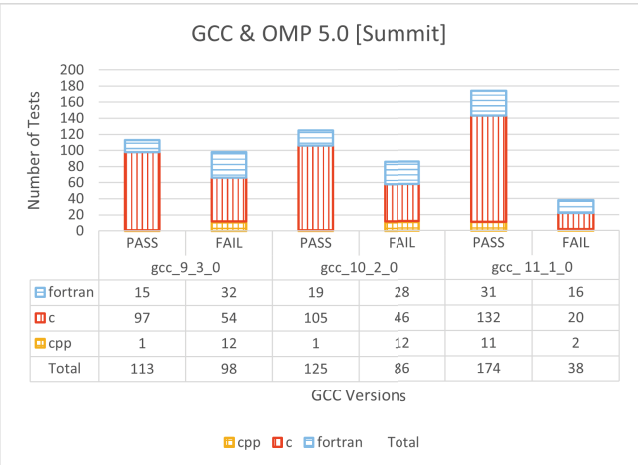
	PASS	FAIL	PASS	FAIL	PASS	FAIL
	llvm_13		llvm_14		llvm_15	
c	114	0	114	0	114	0
cpp	14	0	14	0	14	0
Total	128	0	128	0	128	0

Fig. 2: OpenMP 4.5 tests with LLVM on Summit.



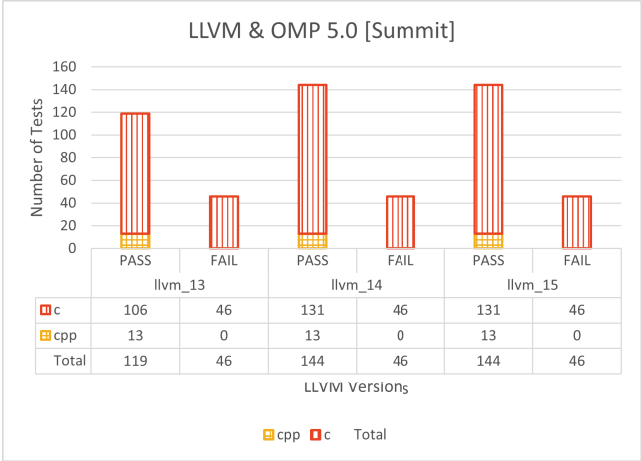
	PASS	FAIL	PASS	FAIL	PASS	FAIL
	nvhpc_21_7		nvhpc_21_9		nvhpc_21_11	
fortran	90	10	90	10	97	3
c	111	3	111	3	112	2
cpp	14	0	14	0	14	0
Total	215	13	215	13	223	5

Fig. 3: OpenMP 4.5 tests with NVHPC on Summit.



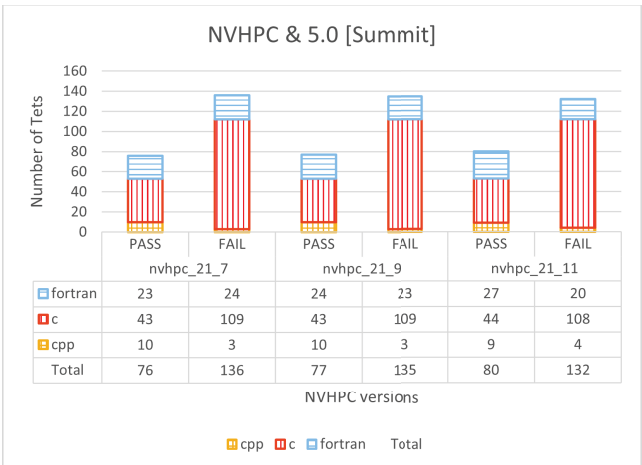
	PASS	FAIL	PASS	FAIL	PASS	FAIL
	gcc_9_3_0		gcc_10_2_0		gcc_ 11_1_0	
fortran	15	32	19	28	31	16
c	97	54	105	46	132	20
cpp	1	12	1	12	11	2
Total	113	98	125	86	174	38

Fig. 4: OpenMP 5.0 tests with GCC on Summit.



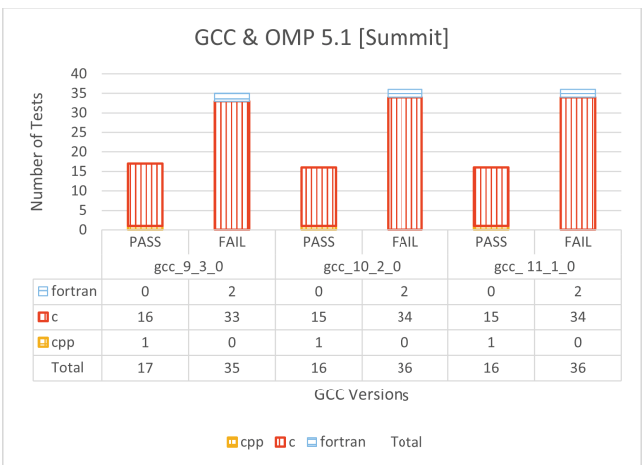
	PASS	FAIL	PASS	FAIL	PASS	FAIL
	llvm_13		llvm_14		llvm_15	
c	106	46	131	46	131	46
cpp	13	0	13	0	13	0
Total	119	46	144	46	144	46

Fig. 5: OpenMP 5.0 tests with LLVM on Summit.



	PASS	FAIL	PASS	FAIL	PASS	FAIL
	nvhpc_21_7		nvhpc_21_9		nvhpc_21_11	
fortran	23	24	24	23	27	20
c	43	109	43	109	44	108
cpp	10	3	10	3	9	4
Total	76	136	77	135	80	132

Fig. 6: OpenMP 5.0 tests with NVHPC on Summit.



	PASS	FAIL	PASS	FAIL	PASS	FAIL
	gcc_9_3_0		gcc_10_2_0		gcc_ 11_1_0	
fortran	0	2	0	2	0	2
c	16	33	15	34	15	34
cpp	1	0	1	0	1	0
Total	17	35	16	36	16	36

Fig. 7: OpenMP 5.1 tests with GCC on Summit.

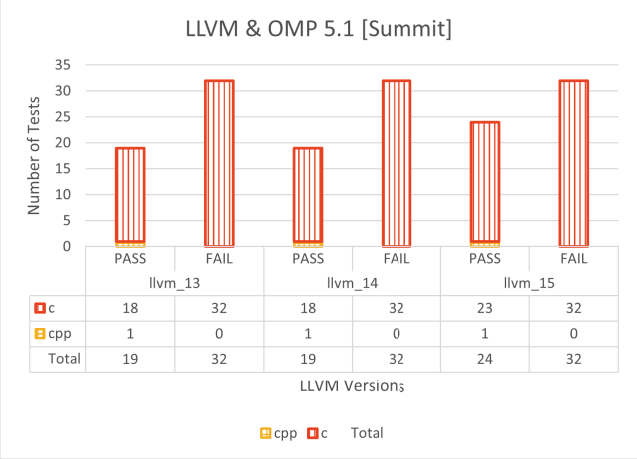


Fig. 8: OpenMP 5.1 tests with LLVM on Summit.

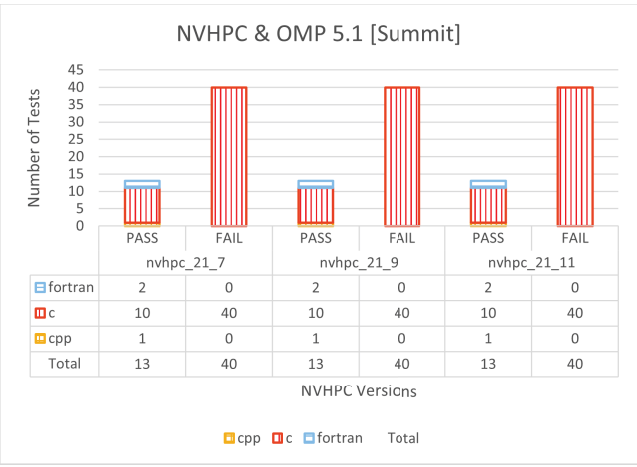


Fig. 9: OpenMP 5.1 tests with NVHPC on Summit.

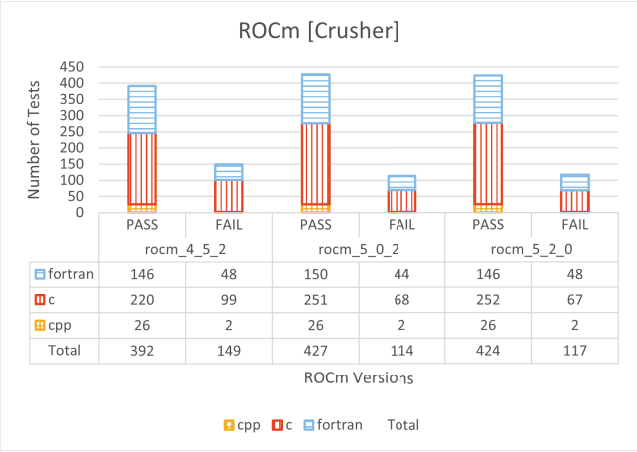


Fig. 10: OpenMP tests with ROCm on Crusher.

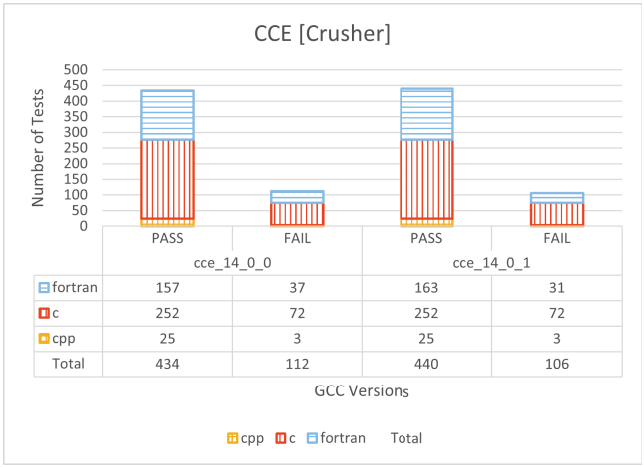


Fig. 11: OpenMP tests with CCE on Crusher.

could be due to features being 'completed' and then reopened for further investigation, becoming 'partial'. Results in Table III show a list of tests that have passed and failed over a set of LLVM compiler versions they have been tested on. For OMP 5.0 test written for `master loop device` passes on LLVM version 13 but fails in the next two versions i.e. 14 and 15. Similar trends are seen for five other OMP 5.0 tests while the OMP 5.0 test for `reverse offload` only fails on LLVM version 15, which is the latest tested version.

Test Name	OMP Ver	gcc 9.3.0	gcc 10.2.0	gcc.11.1.0
test_loop_reduction _and_device.c	5.0	Pass	Fail	Fail
test_loop_reduction _or_device.c	5.0	Pass	Fail	Fail

TABLE II: Inconsistencies of tests passing and failing across different GCC versions

Test Name	OMP Ver	llvm_13	llvm_14	llvm_15
test_master_taskloop _device.c	5.0	Pass	Fail	Fail
test_master_taskloop_ simd_device.c	5.0	Pass	Fail	Fail
test_parallel_master _device.c	5.0	Pass	Fail	Fail
test_parallel_master_ taskloop_device.c	5.0	Pass	Fail	Fail
test_parallel_master _taskloop_simd_device.c	5.0	Pass	Fail	Fail
test_requires_reverse _offload.c	5.0	Pass	Pass	Fail
test_target_task_depend _mutexinoutset.c	5.0	Pass	Fail	Fail

TABLE III: Inconsistencies of tests passing and failing across different LLVM versions

3) NVHPC Maturity Over Time

The results collected in Figures 3, 6 & 9, regarding NVHPC are on OLCF's Summit supercomputer system. We targeted the last three stable releases of the NVIDIA HPC compiler suite

including the latest, 21.11. For these results, it is important to note that while coverage for 4.5 is about complete, acceleration of coverage for 5.0 has not increased quickly over the last few releases. Additionally, only 13 of the 53 features that we have written tests for OpenMP 5.1 for are supported. Results in Table IV show a list of tests that have passed and failed over a set of NVHPC compiler versions they have been tested on. For OpenMP 4.5 version, tests written for `target teams distribute for if parallel modifier` passes using NVHPC version 21.7 but fails in the next two versions i.e. 21.9 and 22.11. Similar trends can be seen with tests in OpenMP version 5.0.

Test Name	OMP Ver	21.7	21.9	21.11
test_target_teams_ distribute_for_ parallel_for_if_ parallel_modifier.c	4.5	Pass	Pass	Fail
lsms_triangular_packing.cpp	5.0	Pass	Pass	Fail
test_declare_variant.F90	5.0	Pass	Pass	Fail
test_target_ teams_distribute_parallel _for_collapse.c	5.0	Pass	Pass	Fail

TABLE IV: Inconsistencies of tests passing and failing across different NVHPC versions

B. Results from Crusher - A Pre-Frontier System

Here we share the evaluation of compiler implementations on Crusher. We evaluate AMD ROCm and Cray CCE compilers.

1) ROCm Maturity Over Time

The results listed in Figure 10 are from the Pre-Frontier Crusher system and show 4 versions of AMD's developmental HPC ROCm compiler. Results show a leap in OpenMP implementation from version 4.5.0 to version 5.0.0, but minimal changes there onward. It is interesting to note that one C test now passed from version 5.1.0, but one Fortran test now fails. This, again, could be due to features requiring more investigation or the definition of features being changed in the newer versions of OpenMP.

2) CCE Maturity Over Time

The results listed in Figure 11 also show the Cray Compiling Environment (CCE) results on Crusher. The results only include CCE 14.0.0 & CCE 14.0.1 versions, as the only other versions available on Crusher, 13.0.0 & 13.0.2 do not work properly with OpenMP. These versions require dependencies from both ROCm 4 & ROCm 5, which cannot be loaded at the same time. The results show decent performance, with around 80% of tests passing for version 14.0.0, increasing slightly with 5 more Fortran tests passing in 14.0.1. It is interesting to note that C implementation for CCE & ROCm compiler is nearly identical, but Fortran implementation on CCE is slightly better.

C. Subset of OpenMP Supported By All Compilers

The run-time and compile-time results for NVIDIA, LLVM, GNU, CCE, and ROCm on the Summit and Crusher supercomputing systems reveal a subset of OpenMP features supported by all of the aforementioned compilers. Also revealed, is a subset of OpenMP features not supported by any of the aforementioned compilers. In this section's discussion, a pass is only deemed a pass if it is such for each of the five compilers. Thus, a fail is only deemed a fail if it is such for each of the five compilers. Regarding discussion of Fortran in this section, we will be analyzing results for NVIDIA, GNU, CCE, and ROCm exclusively, as LLVM does not have support for OpenMP offloading on the Summit supercomputer. This point will be belabored in order to avoid any misrepresentation. For our C/C++ OpenMP 4.5 tests, we report 85.93% tests pass and 0% of tests fail across all five compilers. Moreover, for the Fortran OpenMP 4.5 tests, 74.50% of these tests pass and 0% of tests fail across GNU, NVIDIA, CCE, and ROCm. These results exemplify OpenMP 4.5's maturity as almost all of the OpenMP 4.5 features that tested portable across all of the five compilers for C/C++ and four of the compilers for Fortran. Equally as impressive is the fact 0% of tests fail across the aforementioned compilers for C/C++ and Fortan, meaning that every feature tested in OpenMP 4.5 is supported by at least one compiler. Moving onward to OpenMP 5.0, we report 18.34% of the C/C++ tests pass and 4.14% fail across all five compilers. For the Fortran OpenMP 5.0 tests, we report a 16% pass rate and a 23% fail rate across NVIDIA and GNU. While OpenMP 5.0 is certainly not as portable as OpenMP 4.5, the results expose the fact that almost every OpenMP 5.0 feature we test is supported by at least one compiler. For the C/C++ OpenMP 5.1 tests, only 1.96% of tests pass and 43.13% of tests fail for C/C++ tests pass across all three compilers. Finally, the Fortran OpenMP 5.1 tests have a 0% pass rate and a 50% fail rate across NVIDIA and GNU.

V. Impacting OpenMP Community, Vendors, OpenMP Specification, and Applications

This section draws inferences from the lessons learnt via this project and highlights the impact of SOLLVE V&V on on the different aspects related to OpenMP; mainly community, vendors, specification, and applications.

A. Impact on the OpenMP Community

With rapid development of the OpenMP Specification and with OpenMP Examples [5] as the only OpenMP ARB sanctioned resource, application programmers and other users often refer to the SOLLVE V&V tests to see how to utilize a new OpenMP features. One of reasons for this is that OpenMP Examples [5], though an excellent resource, is not exhaustive and is often released after a given specification version has been around for sometime. The SOLLVE V&V also consists of some tests adapted from OpenMP Examples document, but recently, the examples document has been extended to include an `declare target` example based on the SOLLVE V&V feature test for `device_type(nohost)` based on the community discussion during the test creation. Listing 4 shows the skeleton testcase. Due to the fundamental OpenMP requirement that fallback execution of device constructs must be supported, using `device_type(nohost)` for procedure

target_fun on the `declare target` imposes an additional requirement not clearly mentioned in the specification. Since `nohost` implies that the procedure *target_fun* is made available only on the device, it needs to be a device variant for the procedure *fun()*. This is needed to ensure that a host symbol for *target_fun* is not required to be present in the host environment in the case of host fallback. Without the variant function, the use of `nohost` will result in a link time error due to the code generated for host execution of the target region.

```
...
#pragma omp declare variant(target_fun) match(
    device={kind(nohost)})
void fun() {
  /*some work*/
} }
void target_fun(){
  /*some work*/
}
#pragma omp declare target enter(target_fun)
    device_type(nohost)
int main() {
  ...
  #pragma omp target
  {
    foo(); // calls the target_fun() on device or
     fun() in case of host fallback.
  }
  ...
```

Listing 4: Snippet from declare target directive test with device_type nohost

B. Impact on Vendor Implementation

The test cases we create can also help expose missing aspects of a specific compiler's implementation. In one instance https://github.com/SOLLVE/sollve_vv/issues/409, we were developing a test to evaluate the new `metadirective` feature. The objective of the test was to check the use of context-selectors to determine which vendor provided the implementation, either AMD or NVIDIA in this case. Then, depending on which vendor produced the current implementation, we would run with a different number of threads, 32 for NVIDIA or 64 for AMD. After approving and merging this test, a developer from NVIDIA noticed that we had incorrectly used an `omp_is_initial_device` runtime call strictly nested inside of a teams region as shown below.

Although this test was not for the teams directive, nor for `omp_is_initial_device`, this mishap led GCC to add in additional API call checks for constructs strictly nested inside teams https://gcc gnu org/PR102972

```
#pragma omp metadirective \
     when( implementation=vendor(nvidia): \
        teams num_teams(512) thread_limit(32) ) \
    when( implementation=vendor(amd): \
        teams num_teams(512) thread_limit(64) ) \
    default (teams)
      which_device = omp_is_initial_device();
      #pragma omp distribute parallel for
         for (i = 0; i < N; i++) {
            a[i] = i;
         }
```

Listing 5: Incorrectly Strictly Nested OpenMP runtime call

C. Changes to the OpenMP 6.0 Specification

1) Discussion of Test Case Leads to Specification Issue

In another instance, a line of questioning regarding one of our already peer reviewed and merged pull requests that came in the form of a GitHub issue led to discussion with the OpenMP Language Committee. The issue, also described here https://github.com/SOLLVE/sollve_vv/issues/426 and shown in the code caption below, pointed out a unique case where a local variable is mapped to the device using a `target enter data map`, but is not explicitly mapped again on the target region itself. The stack variable is then treated as `firstprivate` in the target region and is not deallocated properly causing the stack address to be reused by a different stack variable. In this case, confusion arises due to discrepancies in the present table and produces a runtime error. The fix we agreed upon with the community member who discovered this issue is to free memory on the device associated with the stack variables before the lifetime of said variable ends on the host. Even though we were able to resolve this runtime error through deallocating the variable at the proper time, it became clear that there is no wording in the OpenMP specification that states that an OpenMP programmer must use a `target exit data` or similar directive to ensure that the lifetime of a variable does not end before it has been unmapped from a device data environment. An issue was filed with the OpenMP specification for inclusion in the 6.0 specification, but has not been resolved or merged yet.

```
#pragma omp target enter data map(to: val) depend
    (out: val)

#pragma omp target map(tofrom: isHost) map(alloc:
     h_array[0:N]) depend(inout: h_array) depend(
    in: val)
  {
    isHost = omp_is_initial_device();
    for (int i = 0; i < N; ++i) {
      h_array[i] = val; // val = DEVICE_TASK1_BIT
    }
  }
```

Listing 6: Confusion surrounding lifetime of stack variable var

2) Specification Clarification from Test Case Discussion

Further success resulted from our test case of the recently added `allocate` directive https://github.com/SOLLVE/sollve_vv/pull/440. An in-depth discussion regarding whether a certain variable could or could not be explicitly mapped, led to the inclusion of the following language in the restrictions of the threadprivate directive: "A variable that is part of another variable (as an array element or a structure element) may appear in a threadprivate directive only if it is a static data member of a C++ class."

D. Impact on Applications

Though the SOLLVE V&V tests primarily focus on feature tests, it also covers certain pure OpenMP application kernels extracted from applications. These tests focus on particular requirement from applications or motivated by inconsistent, regressions seen in vendor implementation, and varying support from vendors. For example, the QMCPACK application

(an ECP application that is open-source, high-performance electronic structure code that implements numerous Quantum Monte Carlo (QMC) algorithms [12]) found out that certain OpenMP implementations would error out when Math library functions (in `math.h`) were invoked from within a `target` region. In order to simplify testing over multiple implementations we created a distilled version of the same and included in our *application_kernels* directory. The developers can run these tests on the platform of their choice or refer to the SOLLVE V&V website [25] to see if recent results for the same to verify support. The SOLLVE V&V is tested across a variety of hardware platforms and OpenMP implementations regularly to document the progress of the different implementations and record any regressions. Likewise we have looked into a number of applications like GridMINI, GESTS, LSMS etc. to collect representative application kernels which are of interest for the application developers to track. These kernels not only provide an quick and easy method for application developers to check vendor implementations, but also provide insights to vendors regarding the pain-points or critical features required by HPC applications.

VI. Related Work

Work on OpenMP offloading has evolved in the past several years. Updated information on the various compiler tools and their coverage of OpenMP implementations especially offloading features can be found here [19].

Following are some of related works on the validation and verification of OpenMP implementations that includes features prior to offloading as well [8], [9], [15], [16], [26]. These works have been highlighting ambiguities in the specifications and reporting compiler/runtime bugs thus enabling application developers to be aware of the status of the compilers. Another effort to test OpenMP Offloading test functions for C++ and Fortran is the OvO suite [3]. These tests focus on extensively testing hierarchical parallelism and mathematical functions.

Other related work includes Csmith [27], a comprehensive, well-cited work where the authors perform a randomized test-case generator exposing compiler bugs using differential testing. Csmith detects compiler bugs, however the strategy entails automatically mapping a randomly generated failed test to a bug that actually caused it. Such a strategy would be effective on implementations that are stable and mature. However in our case, there is frequent communication with vendors with respect to discussing and reporting bugs, and the suite also requires the use of combined and composite directives that need to be tested prior to marking a bug as a compiler or a runtime error. To that end the testsuite is not quite ready to use an approach like that used in Csmith.

Other related work includes the parallel testsuite [10] that chooses a set of routines to test the strength of a computer system (compiler, runtime system, and hardware) in a variety of disciplines with one of the goals being to compare the ability of different Fortran compilers to automatically parallelize various loops. The Parallel Loops test suite is modeled after the Livermore Fortran kernels [14]. Overheads due to synchronization, loop scheduling and array operations are measured for the language constructs used in OpenMP in [22]. Significant differences between the implementations are observed, which suggested possible means of improving future performance. A microbenchmark [6] suite was developed to measure the overhead of the task construct introduced in the OpenMP 3.0 standard, and associated task synchronization constructs.

These above mentioned work are some of the closely related work that focuses on tests being built and measuring overheads of implementations. There are several other related efforts that evaluate implementations using proxy, mini- or real-world applications. These work focus on mostly for performance evaluation and not the validity of the implementations. Some of these work include [7], [11], [13], [21].

VII. Conclusion

Application developer teams often use OpenMP to improve the performance of their code. Newer versions of OpenMP are released every other year which include GPU offloading features, and it is vital that these features are implemented by compiler vendors & system managers. Our testsuite ensures developers know what systems & compilers perform the most optimally for C, C++ & Fortran.

The test suite has been run on multiple systems, including ORNL's Summit system and the Pre-Frontier systems Crusher with multiple compilers. Overall, it is obvious GCC, Clang & NVHPC perform similarly for OpenMP 4.5 features, while NVHPC falls behind in later versions. LLVM's lack of a Fortran compiler makes it difficult to compare these compilers as a whole, though. On Crusher, which uses an AMD GPU, both ROCm and CCE have better support but do not progress much over version releases, especially regarding Fortran support.

Analysis of the suite's results for NVIDIA, LLVM, GNU, CCE, and ROCm on the Summit and Crusher supercomputing has shown that OpenMP 4.5 is extremely portable across the five compilers observed in this study. OpenMP 5.0 is not very portable across these compilers and some features are not supported by any of the five compilers. Developers that are concerned about portability are recommended to utilize OpenMP 4.5 features and only utilize OpenMP 5.0 features that are in the sparse set of features supported by all compilers.

Despite challenges presented to test writing, compiler implementation continues to improve over time and newer versions of OpenMP feature tests, presently supporting OpenMP 5.2, are being included in our testsuite.

VIII. Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. The research is also supported by the NSF under grant no. 1814609.

REFERENCES

[1] S. Abbot. https://www.openmp.org/wp-content/uploads/2022-04-29-ECP-OMP-Telecon-HPE-Compiler.pdf.

[2] AMD. HIP-Supported CUDA API Reference Guide v 4.5. [Online]. Avaialble: https://github.com/RadeonOpenCompute/ROCm/blob/rocm-4.5.2/AMD_HIP_Supported_CUDA_API_Reference_Guide.pdf.

[3] T. Applencourt. https://github.com/TApplencourt/OvO.

[4] D. Black. Frontier named no. 1 supercomputer on top500 list and 'first true exascale machine'. https://insidehpc.com/2022/05/frontier-named-no-1-supercomputer-on-top500-list-and-first-true-exascale-machine/.

[5] O. A. R. Board. OpenMP Application Programming Interface Examples. [Online]. Available: https://www.openmp.org/wp-content/uploads/openmp-examples-5-2.pdf, 2022.

[6] J. M. Bull, F. Reid, and N. McDonnell. A microbenchmark suite for openmp tasks. In *International Workshop on OpenMP*, pages 271–274. Springer, 2012.

[7] J. H. Davis, C. Daley, S. Pophale, T. Huber, S. Chandrasekaran, and N. J. Wright. Performance assessment of openmp compilers targeting nvidia v100 gpus. In *International Workshop on Accelerator Programming Using Directives*, pages 25–44. Springer, 2020.

[8] J. M. Diaz, K. Friedline, S. Pophale, O. Hernandez, D. E. Bernholdt, and S. Chandrasekaran. Analysis of openmp 4.5 offloading in implementations: correctness and overhead. *Parallel Computing*, 89:102546, 2019.

[9] J. M. Diaz, S. Pophale, K. Friedline, O. Hernandez, D. E. Bernholdt, and S. Chandrasekaran. Evaluating support for openmp offload features. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, page 31. ACM, 2018.

[10] J. Dongarra, M. Furtney, S. Reinhardt, and J. Russell. Parallel loops?a test suite for parallelizing compilers: Description and example results. *Parallel Computing*, 17(10-11):1247–1255, 1991.

[11] R. Gayatri, C. Yang, T. Kurth, and J. Deslippe. A case study for performance portability using openmp 4.5. In *International Workshop on Accelerator Programming Using Directives*, pages 75–95. Springer, 2018.

[12] P. R. C. Kent, A. Annaberdiyev, A. Benali, M. C. Bennett, E. J. Landinez Borda, P. Doak, H. Hao, K. D. Jordan, J. T. Krogel, I. Kylänpää, J. Lee, Y. Luo, F. D. Malone, C. A. Melton, L. Mitas, M. A. Morales, E. Neuscamman, F. A. Reboredo, B. Rubenstein, K. Saritas, S. Upadhyay, G. Wang, S. Zhang, and L. Zhao. Qmcpack: Advances in the development, efficiency, and application of auxiliary field and real-space variational and diffusion quantum monte carlo. *The Journal of Chemical Physics*, 152(17):174105, 2020.

[13] M. Khalilov and A. Timoveev. Performance analysis of cuda, openacc and openmp programming models on tesla v100 gpu. In *Journal of Physics: Conference Series*, volume 1740, page 012056. IOP Publishing, 2021.

[14] F. H. McMahon. The livermore fortran kernels: A computer test of the numerical performance range. Technical report, Lawrence Livermore National Lab., CA (USA), 1986.

[15] M. Müller and P. Neytchev. An openmp validation suite. In *Fifth European Workshop on OpenMP, Aachen University, Germany*. Citeseer, 2003.

[16] M. S. Müller, C. Niethammer, B. Chapman, Y. Wen, and Z. Liu. Validating openmp 2.5 for fortran and c/c++. In *Sixth European Workshop on OpenMP, KTH Royal Institute of Technology, Stockholm, Sweden*, 2004.

[17] NVIDIA, P. Vingelmann, and F. H. Fitzek. Cuda, release: 10.2.89. [Online]. Available: https://developer.nvidia.com/cuda-toolkit, 2020.

[18] OpenMP. OpenMP API Specification: Version 5.1 November 2020. [Online]. Available: https://www.openmp.org/spec-html/5.1/openmpsu68.html, 2020.

[19] OpenMP. OpenMP compilers and tools. [Online]. Available: https://www.openmp.org/resources/openmp-compilers-tools/, 2022.

[20] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 3.0. [Online]. Available: http://www.openmp.org/mp-documents/spec30.pdf, May 2008.

[21] S. J. Pennycook, J. D. Sewall, and J. R. Hammond. Evaluating the impact of proposed openmp 5.0 features on performance, portability and productivity. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 37–46. IEEE, 2018.

[22] F. J. Reid and J. M. Bull. OpenMP microbenchmarks version 2.0. In *Proc. EWOMP*, pages 63–68, 2004.

[23] Top500. June 2022 — top 500. https://www.top500.org/lists/top500/2022/06/.

[24] Top500. Supercomputer fugaku, supercomputer fugaku, a64fx 48c 2.2ghz, tofu... https://www.top500.org/system/179807/.

[25] S. Validation and V. Team. https://crpl.cis.udel.edu/ompvvsollve/results/.

[26] C. Wang, S. Chandrasekaran, and B. Chapman. An openmp 3.1 validation testsuite. In *International Workshop on OpenMP*, pages 237–249. Springer, 2012.

[27] X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in c compilers. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, pages 283–294, 2011.

APPENDIX

A. Abstract

This paper explores the conformity and implementation progress of various compilers for the OpenMP 4.5, 5.0 and 5.1 release specifications. Various scripts were built for testing the implementation, and to gather results from various systems. This repository includes information on the resources such as scripts, hardware, software and other dependencies that can be used to reproduce the results that are being used in the paper.

B. Artifact Availability

Software Artifact Availability: All software is maintained in an repository under Open-Source License BSD-3.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All data, except Crusher results are available on our website and is under Open-Source License BDS-3. Crusher results are under the discretion of OLCF.

Proprietary Artifacts: There are no author-created propritary artifacts.

List of URLs and/or DOIs where artifacts are available:
https://github.com/SOLLVE/sollve_vv
https://crpl.cis.udel.edu/ompvvsollve

C. Baseline experimental setup, and modifications made for the paper

1) Summit

Relevant hardware details: [2x] IBM's 22 SIMD Multi-Core POWER9 CPUs, 512 GB of DDR4, [6x] NVIDIA Tesla V100

Operating systems and versions: Red Hat Enterprise Linux (RHEL) version 8.2

Compilers and versions: GCC 11.2.0 IBM XL 16.1.1-10

Applications and versions: CUDA 11.5.2

Libraries and versions: OpenMP 4.5, 5.0, & 5.1

Paper Modifications: No modifications were made

2) Crusher

Relevant hardware details: 64-core AMD EPYC 7A53 CPU, 512 GB of DDR4, [4x] AMD MI250X

Operating systems and versions: SUSE Linux Enterprise Server 15.3 SP3

Compilers and versions: Cray CCE 14.0.0 & 14.0.01, AMD ROCm 4.5.0, 5.0.0, 5.1.0 & 5.2.0

Applications and versions: No applications were used

Libraries and versions: OpenMP 4.5, 5.0 & 5.1

Paper Modifications: No modifications were made

D. Summit results generation script

```
#!/bin/bash

#Load GCC
module load gcc/11.2.0
module cuda
module python

#run testsuite for 4.5
make CC=gcc CXX=g++ FC=gfortran LOG_ALL=1 LOG=1
     VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=nvidia
     SYSTEM=summit OMP_VERSION=4.5 all

#run testsuite for 5.0
make CC=gcc CXX=g++ FC=gfortran LOG_ALL=1 LOG=1
     VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=nvidia
     SYSTEM=summit OMP_VERSION=5.0 all

#run testsuite for 5.1
make CC=gcc CXX=g++ FC=gfortran LOG_ALL=1 LOG=1
     VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=nvidia
     SYSTEM=summit OMP_VERSION=5.1 all

#Load Clang
module use /sw/summit/modulefiles/ums/stf010/Core
module load llvm/15.0.0-20220420 #might need to
     change this version :)
module load cuda

#run testsuite for 4.5
make CC=clang CXX=clang++ FC=flang LOG_ALL=1 LOG
     =1 VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=
     nvidia SYSTEM=summit OMP_VERSION=4.5 all

#run testsuite for 5.0
make CC=clang CXX=clang++ FC=flang LOG_ALL=1 LOG
     =1 VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=
     nvidia SYSTEM=summit OMP_VERSION=5.0 all

#run testsuite for 5.1
make CC=clang CXX=clang++ FC=flang LOG_ALL=1 LOG
     =1 VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=
     nvidia SYSTEM=summit OMP_VERSION=5.1 all

#Load ibm
module load xl/16.1.1-10
module load cuda

make CC=xlc CXX=xlc++ FC=xlf_r LOG_ALL=1 LOG=1
     VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=nvidia
     SYSTEM=summit OMP_VERSION=4.5 all

#run testsuite for 5.0
make CC=xlc CXX=xlc++ FC=xlf_r LOG_ALL=1 LOG=1
     VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=nvidia
     SYSTEM=summit OMP_VERSION=5.0 all

#run testsuite for 5.1
make CC=xlc CXX=xlc++ FC=xlf_r LOG_ALL=1 LOG=1
     VERBOSE=1 VERBOSE_TESTS=1 DEVICE_TYPE=nvidia
     SYSTEM=summit OMP_VERSION=5.1 all

make report_summary
make report_json
mv report_json summit_results.json
```

Listing 7: Summit script

E. Crusher result generation commands

```
#Load rocm
ml rocm
#Load cray
ml PrgEnv-cray

#run testsuite for cce/14.0.0
ml cce/14.0.0
make CC=cc CXX=CC FC=ftn LOG=1 LOG_ALL=1
     OMP_VERSION=4.5 VERBOSE=1 VERBOSE_TESTS=1
     SYSTEM=crusher DEVICE_TYPE=amd all
make CC=cc CXX=CC FC=ftn LOG=1 LOG_ALL=1
     OMP_VERSION=5.0 VERBOSE=1 VERBOSE_TESTS=1
     SYSTEM=crusher DEVICE_TYPE=amd all
make CC=cc CXX=CC FC=ftn LOG=1 LOG_ALL=1
     OMP_VERSION=5.1 VERBOSE=1 VERBOSE_TESTS=1
     SYSTEM=crusher DEVICE_TYPE=amd all
```

```

#run testsuite for cce/14.0.1
ml cce/14.0.1
make CC=cc CXX=CC FC=ftn LOG=1 LOG_ALL=1
     OMP_VERSION=4.5 VERBOSE=1 VERBOSE_TESTS=1
     SYSTEM=crusher DEVICE_TYPE=amd all
make CC=cc CXX=CC FC=ftn LOG=1 LOG_ALL=1
     OMP_VERSION=5.0 VERBOSE=1 VERBOSE_TESTS=1
     SYSTEM=crusher DEVICE_TYPE=amd all
make CC=cc CXX=CC FC=ftn LOG=1 LOG_ALL=1
     OMP_VERSION=5.1 VERBOSE=1 VERBOSE_TESTS=1
     SYSTEM=crusher DEVICE_TYPE=amd all

#run testsuite for rocm/4.5.0
module load PrgEnv-amd
module load rocm/4.5.0
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=4.5 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.0 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.1 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all

#run testsuite for rocm/5.0.0
module load rocm/5.0.0
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=4.5 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.0 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.1 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all

#run testsuite for rocm/5.1.0
module load rocm/5.1.0
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=4.5 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.0 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.1 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all

#run testsuite for rocm/5.2.0
module load rocm/5.2.0
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=4.5 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.0 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
make CC=amdclang CXX=amdclang++ FC=ftn LOG=1
     LOG_ALL=1 OMP_VERSION=5.1 VERBOSE=1
     VERBOSE_TESTS=1 SYSTEM=crusher DEVICE_TYPE=
     amd all
```

Listing 8: Crusher Commands

F. Sample Result Output

Shown below is a single test result sampled from the results json file generated by the Crusher results generation commands

```
{
"Binary path": "bin/alpaka_complex_template.cpp",
"Compiler command": "amdclang++ -I./ompvv -std=c
    ++11 -lm -O3 -fopenmp -fopenmp -fopenmp-
    targets=amdgcn-amd-amdhsa -Xopenmp-target=
    amdgcn-amd-amdhsa -march=gfx90a  -
    D__NO_MATH_INLINES -U__SSE2_MATH__ -
    U__SSE_MATH__",
"Compiler ending date": "Thu 14 Jul 2022 04:30:15
     PM EDT",
"Compiler name": "amdclang++ AMD clang version
    13.0.0 (https://github.com/RadeonOpenCompute/
    llvm-project roc-4.5.0 21422
    e2489b0d7ede612d6586c61728db321047833ed8)",
"Compiler output": "",
"Compiler result": "PASS",
"Compiler starting date": "Thu 14 Jul 2022
    04:30:03 PM EDT",
"OMP version": "4.5",
"Runtime ending date": "Thu 14 Jul 2022 04:30:15
    PM EDT",
"Runtime only": false,
"Runtime output": "\u001b[0;32m \n\n running: bin
    /alpaka_complex_template.cpp.run \u001b[0m\
    nalpaka_complex_template.cpp.o: PASS. exit
    code: 0\n\u001b[0;31malpaka_complex_template.
    cpp.o:\n[OMPVV_INFO: alpaka_complex_template.
    cpp:40] Test is running on device.\n[
    OMPVV_INFO: alpaka_complex_template.cpp:58]
    The value of errors is 0.\n[OMPVV_RESULT:
    alpaka_complex_template.cpp] Test passed on
    the device.\u001b[0m\n",
"Runtime result": "PASS",
"Runtime starting date": "Thu 14 Jul 2022
    04:30:14 PM EDT",
"Test comments": "none",
"Test gitCommit": "98cae2b",
"Test name": "alpaka_complex_template.cpp",
"Test path": "tests/4.5/application_kernels/
    alpaka_complex_template.cpp",
"Test system": "crusher"
}
```

Listing 9: Sample results json output