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 Electronic Design Automation
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, plenty of 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, we focus on evaluating the conformity and implementation progress of various compiler vendors such as Cray, IBM, GNU, Clang/LLVM, NVIDIA, and Intel. We specifically address the 4.5, 5.0, and 5.1 versions of the specification. For our experimental setup, we use the Crusher and Summit computing systems hosted by Oak Ridge National Lab's Computing Facilities. The effort of fault-finding in these implementations is especially valuable for application developers who are using new OpenMP features to accelerate their scientific codes. We present insights into the current implementation status of various vendors, the progression of specific compiler's support for OpenMP over-time, 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, we aim 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 researchers and scientists.

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

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.

I. Introduction

Seven out of the ten fastest supercomputers in the world are heterogeneous systems [21]. Heterogeneous systems may be compromised 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 [22] 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) [2], we again see 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, many programming models and base

languages are now trying to incorporate parallelism that can effectively use the CPU as well as the GPUs. For a long time CUDA [15] 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 [1]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.

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 [18] 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 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 is one component of this software stack. More features that are valuable to developers continue to be added to the OpenMP specification.

As of May 2022, the compilers that offer support for OpenMP offloading features (specification version 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 [17], the only compilers that have any coverage of OpenMP 5.0 offloading features are AMD, GNU, HPE, Intel, LLVM, Siemens, and NVIDIA.

OpenMP 5.0 was released in November 2018 and introduced a wide variety of improvements on heterogeneous target offload and host based features. One 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.

II. Background

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*. OpenMP offers three levels of parallelism (*teams*, *threads*, and *simd lanes*), but typical devices provide only two levels of parallelism; Intel CPUs offer thread and simd level parallelism, NVIDIA GPUs provide thread-block and thread level parallelism, and AMD GPUs provide work-group and work-item level parallelism. Therefore, different OpenMP compilers can choose different parallelism mapping depending on target devices. 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 the simd clause may play an important role when targeting CPUs (mapping OpenMP threads to CPU threads and OpenMP simd lanes to CPU simd lanes).

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.

1) 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 in the 5.0 specification is the 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 architecture is NVIDIA or `when(arch==nvidia)` we can call one 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.

The `declare target` directive, also introduced in 5.0, allows the user to explicitly ensure that procedures and global variables can be accessed on a device. A key functionality of this new directive is allowing 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.

Marching forward, 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.

III. Motivation

The need to validate and verify the compiler implementations of OpenMP features becomes more important than ever as the Frontier supercomputing systems being made available for use by application and software developers. This paper elaborates on the 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.

IV. SOLLVE Validation and Verification Suite

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 essentially 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 we create a priority list of the most desired features from the new features introduced. When writing a new test for an already implemented feature, we start by interpreting the usability of the feature that is outlined by the specification. Then, we outline the potential combinations of options that may be presented, for example the `default` clause, which was introduced in OpenMP 5.1, has options for `firstprivate`, `private`, and `lastprivate`. For this example, we would need to create three tests to encapsulate the full functionality of this new feature. Lastly, we pay careful attention to the 'restrictions' section of each feature, to ensure that the test we are writing does not violate boundaries that have been outlined by the OpenMP ARB. 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 we ensure 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. Once the test is approved by at least two collaborators, it is merged into the main testing suite. A visual flowchart of our workflow is depicted in Figure 1

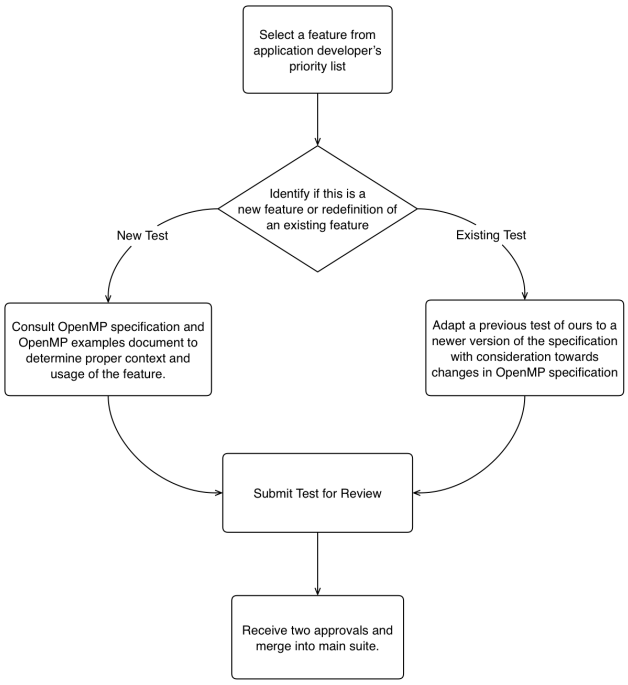

Fig. 1: Process flow demonstrating the typical test creation process.

B. OpenMP 5.x+ Feature Coverage

Currently, our testsuite includes 258 5.0 tests, 45 5.1 tests & 6 5.2 tests. Our 5.0 coverage includes a mix of Fortran & C versions while the majority of our 5.1 tests are coded in C. We have created tests for a vast majority of 5.1 features. Our priorities related to which 5.1 tests we should write first is based not only on which features LLVM has already implemented, but also on the needs of SOLLVE application developers, starting with high-priority, implemented features, and working our way to low-priority non-implemented features. Out of the 13 features listed in the "medium to very-high" range by application teams, 10 have tests written for them. In order to increase our 5.1 coverage, we will continue to create tests in C as a base-line, and afterwards create Fortran versions of these tests. Once our team has written test cases for all of the new features in C and Fortran, our primary focus will shift toward 5.2 features.

C. Challenges

1) Testing Unimplemented Features

Often times we undertake writing test cases 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, with which we may have 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 when we wrote 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.

We initially tested this by looking for any spawned threads, or signs that the array had been changed and thus the code had run despite the `nothing` clause. This seemed promising, but just as the test was beginning to take shape, we discovered our interpretation was not actually how compilers were implementing it. As often times with new features in the specification, explanations can be vague. The `nothing` clause when used outside of the metadirective simply means that OpenMP would ignore the pragma. However, the code would still run in serial. This meant our initial version of the test was wrong and needed to be amended.

Ultimately we reworked the test 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 our example above, that would mean it'd 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 is a perfect example of the challenges of building tests for cutting-edge OpenMP features. Our initial version of a test is not always how the compiler developers interpret the specification's intentions. This is why each and every test is rigorously reviewed to make sure each version fully tests the feature in question. Sometimes, like in this case, review can lead to entire rewrites when new interpretations and implementations surface.

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." [16] While this may seem straight-forward, the purpose of this is relatively unclear. Should these list items be mapped first, and then marked as on the device? If they are already on the device, what is the benefit of listing them there? How do we ensure that the list items are not unmapped at the end of a target region so that we can ensure they are on the target region when utilizing the clause? The difference between `use_device_addr` and `has_device_addr` is not explicitly clear.

Furthermore, this clause was not listed on the OpenMP examples document. [3] This document is often used by our 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: Simple example of has_device_addr directive

Our 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. We 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.

V. 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 2, 3 & 4, 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 is 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.

2) LLVM Maturity Over Time

The results presented in Figures 2, 3 & 4, are for Clang and Clang++ using the LLVM-13 stable release, LLVM-14 stable release, and the LLVM-15 developmental release on OLCF's Summit supercomputer. It is important to note that LLVM provides an `-fopenmp-version` flag that allows you to clarify to 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 could be due to features being 'completed' and then reopened for further investigation, becoming 'partial'.

3) NVHPC Maturity Over Time

The results collected in Figures 2, 3 & 4, 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, none of the 45 features that we have written tests for OpenMP 5.1 for are supported.

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 5 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 being changed in newer versions of OpenMP.

2) CCE Maturity Over Time

The results listed in Figure 5 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

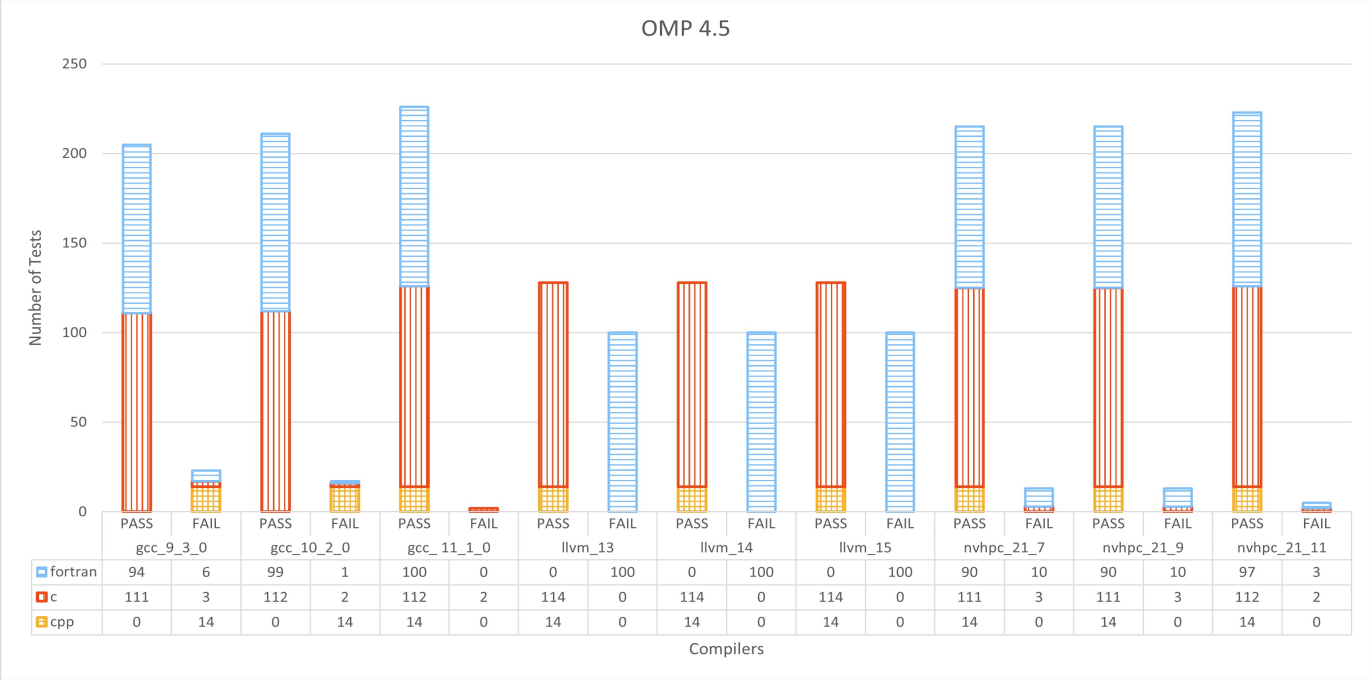

Fig. 2: OpenMP version 4.5 on Summit. This plot shows the tests written in C,C++,Fortran for OpenMP version 4.5 run on Summit using various compilers (GCC,LLVM,NVHPC).These test have been run on three different versions of the compilers listed above GCC(v9.3.0,v10.2.0,v11.1.0) LLVM(v13,v14,v15) and NVHPC(v21.7,v21.9,v21.11) Later versions were not tested due to unavailability. A total of 228 tests were conducted of which 100 are on Fortran, 114 are on C and 14 are on C++ Taking GCC v11.1.0 into account 99.1% of tests pass have passed. The only test that have failed on this compiler are 2 C tests out of the 114 C tests. The greatest improvement is seen across GCC with a pass rate of 89.9% on v9.3.0 improving to 99.1% on v11.1.0. LLVM's performance was stagnant with a pass rate of 56.1% because of all the Fortran tests failing.

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.

VI. Impacting OpenMP through Community Interaction

A. How Our Test Suite Impacts the OpenMP Community

Our team frequently interacts with compiler vendors and open source compiler developers through the means of issues and pull requests on our GitHub. Whether it be including our tests in their continuous integration pipeline, using our tests to validate the specific implementation of an OpenMP feature, or using our test cases to see an example of proper usage a specific feature, we find that some organizations and software developers certainly benefit from the availability and constant maintenance of our test suite. And regardless of the reason for interacting with us, it is apparent that our test suite is one of the only places to find concrete use-case examples for new OpenMP features that have not been implemented yet. Many times we may adapt a test or take inspiration from the OpenMP Examples document, but we would be selling ourselves short if we said that we have not contributed some of the only publicly available use cases of new OpenMP features. When creating a test for a feature that has no easy to find examples online, we run into situations where we may misinterpret the specification. Other times, we run into issues where the specification itself is not clear and could be interpreted many different ways. Either way, putting out a new test often generates discussions amongst people who monitor our test suite, and sometimes this discussion may result in submitting issues to the OpenMP Language Committee.

B. Usage Error leads to Upstream Patch in GCC

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 goal of the test was to check that we can use 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

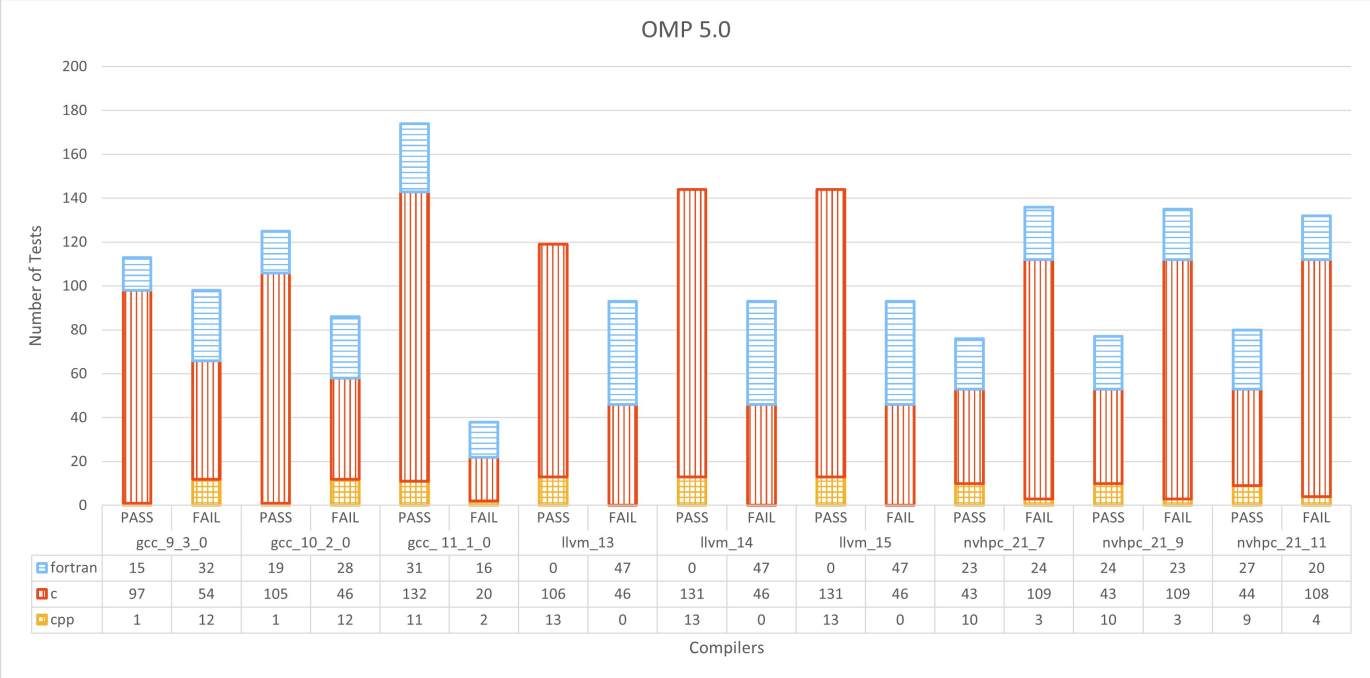

Fig. 3: OpenMP version 5.0 on Summit. This plot shows the tests written in C, C++, Fortran for OpenMP version 5.0 run on Summit using various compilers (GCC,LLVM,NVHPC).These test have been run on three different versions of the compilers listed above GCC(v9.3.0,v10.2.0,v11.1.0) LLVM(v13,v14,v15) and NVHPC(v21.7,v21.9,v21.11). Later versions were not tested due to unavailability. A total of 211 tests were conducted of which 47 are on Fortran, 151 are on C and 13 are on C++. Taking NVHPC v21.9 into account 36.5% of tests pass have passed. The major contributing factor to the low percentage are the 109 C tests that have failed out of the 151 C tests. The greatest improvement is seen across GCC with a pass rate of 53.5% on v9.3.0 improving to 82.4% on v11.1.0.

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 4: 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)
```

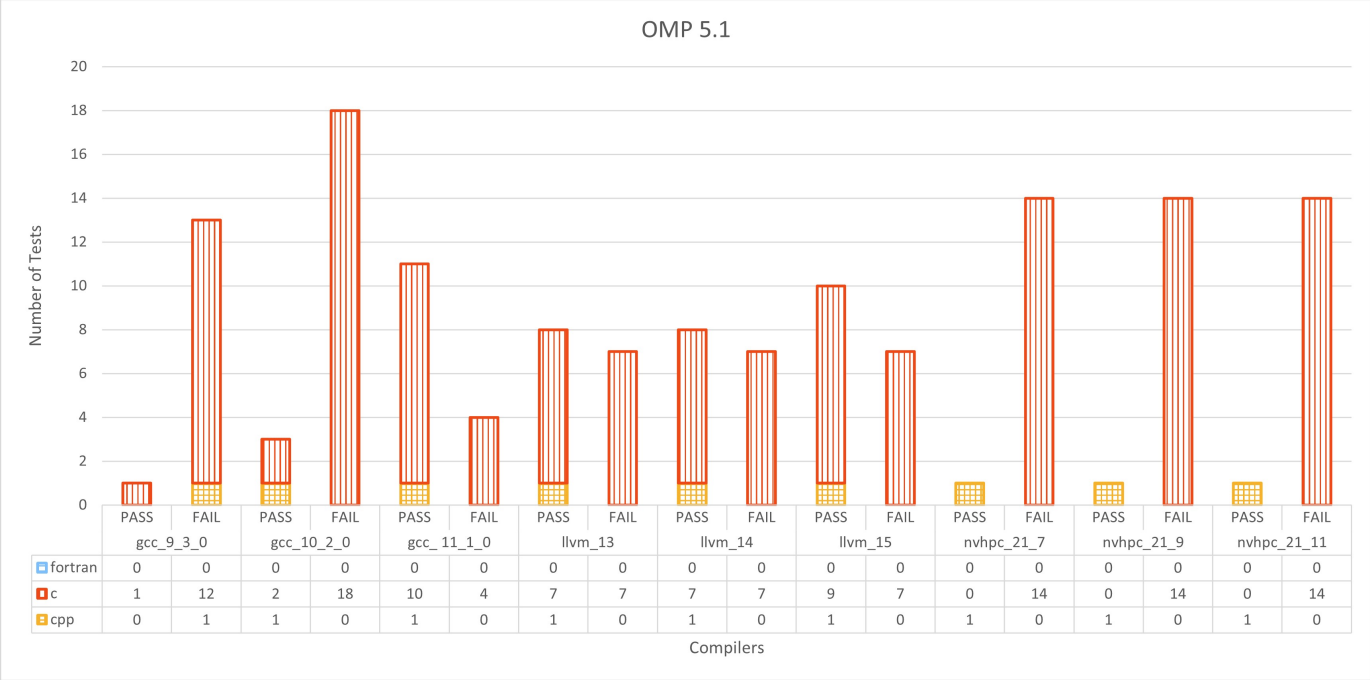

Fig. 4: OpenMP version 5.1 on Summit. This plot shows the tests written in C and C++ for OpenMP version 5.1 run on Summit using various compilers (GCC,LLVM,NVHPC). These test have been run on three different versions of the compilers listed above GCC(v9.3.0,v10.2.0,v11.1.0) LLVM(v13,v14,v15) and NVHPC(v21.7,v21.9,v21.11) Later versions were not tested due to unavailability. Varied number of tests were conducted with the number of C test varying between 13-20 and a single C++ test. Taking LLVM v14 into account 46.6% of tests pass have passed. Half of the 14 C test have failed. The greatest improvement is seen across GCC with a pass rate of 7.1% on v9.3.0 improving to 73.3% on v11.1.0. NVHPC's performance was stagnant with a pass rate of 6.6% because of all the C tests failing.

```
  {
    isHost = omp_is_initial_device();
    for (int i = 0; i < N; ++i) {
      h_array[i] = val; // val = DEVICE_TASK1_BIT
    }
  }
```

Listing 5: Confusion surrounding lifetime of stack variable var

2) New Clarification Introduced due to 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."

3) Potential Change from Test Case Discussion

A unique test we created covered a reduction on the device with two array elements from the same array. This issue is still pending on the OpenMP internal GitHub, and has generated discussion regarding whether allowing this behavior is even plausible since it is likely inefficient. See code listing below.

```

  temps[0] = 0;
  temps[1] = 0;

#pragma omp target map(tofrom: temps)
  {
#pragma omp parallel reduction(+:temps[0], temps
    [1])
    {
      temps[0] += 1;
      temps[1] += 1;
    }
  }
```

Listing 6: Restrictions common to reduction clauses

VII. 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 [17].

Following are some of related works on the validation and verification of OpenMP implementations that includes features prior to offloading as well [6], [7], [13], [14], [23]. 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.

Other related work includes Csmith [24], a comprehensive, well-cited work where the authors perform a randomized test-case generator exposing compiler bugs using differential

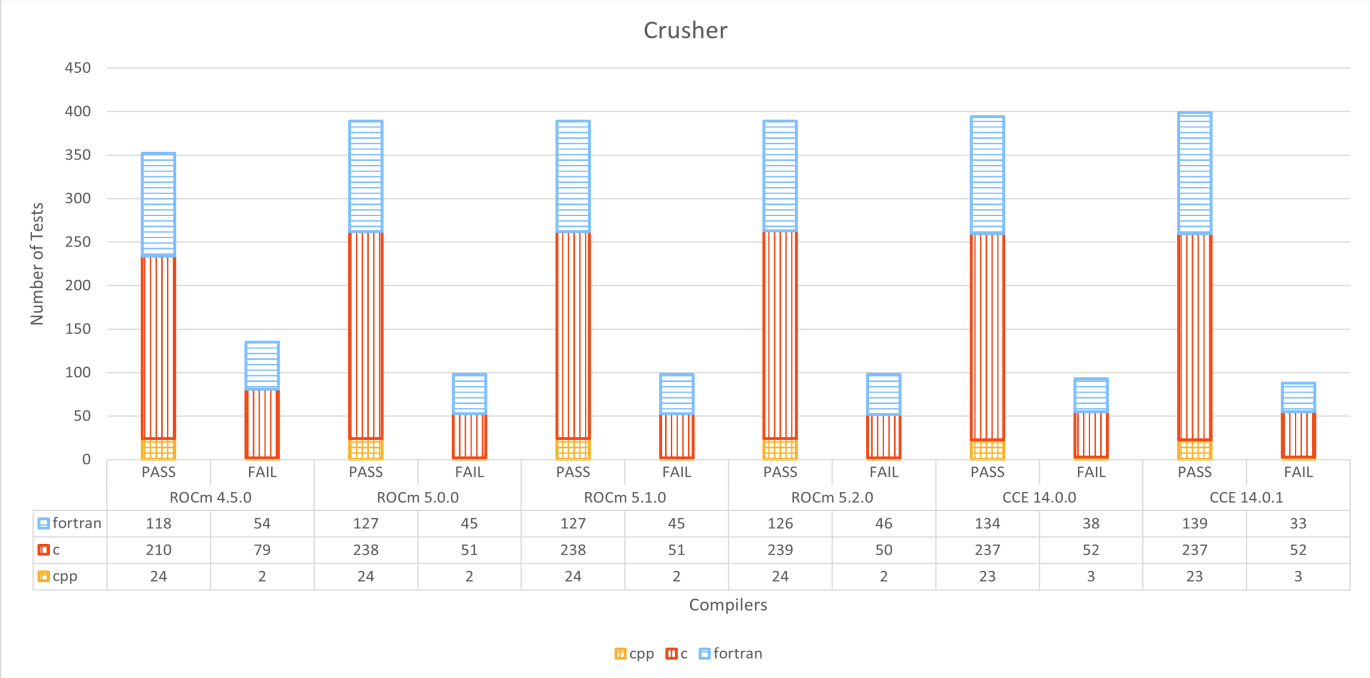

Fig. 5: ROCm and CCE on Crusher. This plot shows the tests written in C, C++, Fortran for OpenMP run on Crusher using ROCm versions v4.5.2, v5.0.0, v5.0.2, v5.1.0, v5.2.0 and CCE versions v14.0.0, v14.0.1. Later versions weren't tested due to unavailability. It is important to note that earlier version of CCE, such as version 13.0.0, weren't able to be tested as an error is present when using the ROCm libamdhip64.so dependency, with both ROCm version 4 & ROCm version 5 dependencies required. A total of 487 tests were conducted, of which 172 are on Fortran, 289 are on C and 26 are on C++. After ROCm version 4.5.2 we see a slight jump in the pass rate, from 72.3% to a constant 79.8% for the following versions. CCE performed well, with 80% pass rate, which slightly increases in CCE 14.0.1 with 5 more Fortran tests passing.

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, we have frequent communication with vendors with respect to discussing and reporting bugs, we also require to use 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 [8] 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 [12]. Overheads due to synchronization, loop scheduling and array operations are measured for the language constructs used in OpenMP in [20]. Significant differences between the implementations are observed, which suggested possible means of improving future performance. A microbenchmark [4] suite was developed to measure the overhead of the task construct introduced in the OpenMP 3.0 standard, and associated task synchronization constructs.

The LLVM open-source compiler infrastructure [11] has a testing setup called *lit* testing tool, which by itself does not contain accelerator (offloading) tests except for a very few tests on offloading and tasking. The LLVM testing infrastructure contains regression tests and whole programs. These regression tests are expected to always pass and should run before every commit. These tests are designed to test the various features of LLVM. Our OpenMP offloading testsuite has been loosely integrated into the LLVM lit infrastructure and we soon plan to tighten up this integration. This way the testsuite can consistently validate and verify OpenMP's offloading features being implemented within LLVM.

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 [5], [9], [10], [19].

VIII. 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. Some newer features, such as the `metadirective` and `nothing` directives, combined with an unclear specification make features difficult to test.

We have ran our suite on multiple systems, including ORNLs' 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 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.

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.

REFERENCES

[1] 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.

[2] 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/.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] LLVM. LLVM Testing Infrastructure Guide. [Online]. Available: http://www.llvm.org/pre-releases/4.0.0/rc2/docs/TestingGuide.html#test-suite.

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

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

[14] 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.

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

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

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

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

[19] 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.

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

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

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

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

[24] 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