OpenMP 4.5 Validation and Verification Suite for Device Offload

Jose Monsalve Diaz[†*], Swaroop Pophale[‡*], Oscar Hernandez[‡], David E. Bernholdt[‡], and Sunita Chandrasekaran[†]
* Contributed Equally

[†]University of Delaware, Newark, DE, USA
{josem, schandra}@udel.edu
[‡]Oak Ridge National Lab, Oak Ridge, TN, USA
{pophaless, oscar, bernholdtde}@ornl.gov

Abstract. OpenMP has been widely adopted for shared memory systems for over a decade. With the heterogeneity trend in architectures rapidly growing, the programming model needed to evolve such that applications could not only be ported to traditional CPUs but also to accelerators often acting as discrete or integrated devices to CPUs. To that end, OpenMP started to provide support for heterogeneous systems since 2013 when the version 4.0 of the specification was ratified. OpenMP 4.5 is being enhanced to cover major requirements of Exascale Computing Project (ECP) applications. As a result it is time-critical to ensure that the implementations of the 4.5 features are correct and conforming to the specification. This paper focuses on building a Validation and Verification testsuite that will test and present results for several offloading features implemented in compilers such as Clang, IBM XL C/C++, CCE, and GCC. We have results for our testsuite on TITAN, Summitdev and Summit at the Oak Ridge National Lab. We will highlight some of the ambiguities we encountered in the process of validating and verifying feature implementations. We also make the testsuite available for anyone to use and will walk the readers through the infrastructure and the workflow of the testsuite. A website has been built to capture our efforts narrated in this paper `https://crpl.cis.udel.edu/ompvvsollve`.

Keywords: OpenMP, Validation and Verification, Testsuite

This manuscript has been co-authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

DRAFT VERSION

1 Introduction

In 2013 the OpenMP specification made a significant shift to provide support for heterogeneous systems. They introduced a set of directives for identifying code as well as data to be moved to the target device for computation. Other than support for accelerators, `SIMD` support for vectorization, thread affinity control, user defined reductions, updates to `task` construct by introducing task groups and dependency clauses were also introduced. This led to the release of Version 4.0. Significant further improvements for device support along with runtime routines for device memory management was introduced in Version 4.5 in 2015 along with new `taskloop` construct that would enable loops to be divided into tasks avoiding the requirement that all threads execute the loop. Support for `doacross` loops to parallelize loops with well-structured dependences were provided. Further support for tasks in the form of task priority was introduced. `SIMD` extensions included the ability to specify exact SIMD width and additional data-sharing attributes.

As the OpenMP specification continues to grow and evolve with all its existing and new features, it is critical to ensure that the different implementations that claim conformance are functionally correct and true to the specification. Having confidence in the correctness of implementations will encourage *users* to adopt OpenMP 4.5 for large applications and port them to heterogeneous systems. Some of the applications that have been ported to GPUs at DOE laboratories using OpenMP 4.5 include Pseudo-Spectral Direct Numerical Simulation-Combined Compact Difference (PSDNS-CCD3D) [1], a computational fluid dynamics code on turbulent flow simulation using GPUs and run to scale on the Titan super-computer, Quick Silver [12], a proxy app of Mercury, that solves a simplified dynamic Monte Carlo particle transport problem. Both these papers discuss the challenges such as heterogeneous memory model, thread safety and thread management, and common programming patterns that are not portable, faced by the application developers before the code ran on a GPU using OpenMP 4.5.

As hardware continues to evolve, the trend for systems to be equipped with specialized accelerators or co-processors attached to a CPU-based system is only going to continue. Top500 reports that a hundred and two systems in the list are configured with accelerators and coprocessors among which, eighty six use NVIDIA GPUs, twelve use Intel Xeon Phi cards, five uses PEZY technology, and two systems use a combination of NVIDIA GPUs and Intel Xeon Phi co-processors [13]. Directive-based programming models, like OpenMP, can prove to be very effective and useful especially when there are legacy codes that need to be migrated to such evolving platforms. Programming models do not require application developers to be fully aware of hardware details or learn newer programming languages thus allowing developers to spend more time on the algorithms and the scientific findings and lesser time on deciphering the intricate details of hardware and language.

Going forward we already know that ORNL's Summit will be a heterogeneous platform consisting of IBM Power9 and NVIDIA's Volta GPUs. We also know

DRAFT VERSION

that Argonne National Lab will receive an Intel-based system (not Knights Hill), Aurora, in the time frame of 2021. Having all this compute power is useless unless we have vetted OpenMP 4.5 implementations that deliver what the specification promises.

Compilers [11] that support OpenMP features include GCC 7.1 where OpenMP 4.5 is fully supported for C and C++. IBM XL C/C++ for Linux V13.1.5 on little endian distributions and XL Fortran for Linux V15.1.15 on little endian distributions (available in Dec 2016) support OpenMP 3.1 and features in OpenMP 4.5 (include device constructs for offloading to NVIDIA GPU), Intel 17.0 supports OpenMP 4.5 for C/C++ and Fortran, Cray Compiling Environment (CCE) 8.5 (June 2016) supports OpenMP 4.0, with OpenMP 4.5 support for device constructs, LLVM Clang 3.9 release supports all non-offloading features of OpenMP 4.5. Clang that supports offloading features is currently in development.

Such a wide range of compilers and interpretation of different implementers can lead to differing implementations of OpenMP features. Our previous publications have captured such discrepancies [6, 15]. To that end, this paper continues to leverage our previous contributions on Validation and Verification testsuite for OpenMP and build a number of functional test cases along with use cases to test 4.5 offloading constructs and clauses or combinations of occurrences of clauses on constructs in order to check for correctness and conformance of features specifically to the 4.5 specification. Once our implementation of the new OpenMP 4.5 features is complete (this is currently on-going work and not fully complete yet), we plan to tie in the 3.1 testsuite and make them all available under one repository to enable testing the entire 4.5 Specification.

This paper makes the following contributions:

- Releases a Validation and Verification testsuite currently comprising of approximately 60 tests that includes unit tests covering extensively target, target data, target enter and exit data features, target update, target teams distribute, and target teams distribute parallel for. We will continue to add more test cases to this suite.
- The testsuite also contains use cases that represent kernels extracted from production DOE applications and other frequently used computation kernels.
- Describe the testsuite infrastructure and add relevant license thus allowing anyone to contribute and to the testsuite.
- Present bugs identified and their potential workarounds thus informing application developers of challenges using key constructs.
- Evaluate implementations of different compilers and verify their conformation to 4.5 specification (these results are posted on `https://crpl.cis.udel.edu/ompvvsollve`).
- List ambiguities in the specification that we came across while interpreting definitions of clauses and constructs and relevant steps taken to clarify them.
- Assemble a user-friendly website with easy to find status update on compiler implementations among other details.

The remainder of the paper is organized as follows: Section 2 discusses some of the most relevant testsuite work by the authors and others. Section 3 gives

DRAFT VERSION

more insight into the new 4.x features. Section 4 explains the workflow and the process of developing such a suite. Section 5 explains the testsuite infrastructure. Section 6 and Section 7 present discussions, conclusion and future work.

2 Related Work

Related work on OpenMP Validation and Verification suite includes [6, 7] that present validation of OpenMP 2.0 implementations which was further extended and improved in [15] to develop a more robust testsuite and provide up-to-date test cases covering all the features until OpenMP 3.1. Some of our other efforts include an OpenACC Validation and Verification testsuite [2,16] where we discuss the compiler status of OpenACC 2.0 and 2.5 specifications respectively and present ambiguities identified in the specification. Some of the tests also focus on challenges with creating unit test cases covering possible combinations of directives/clauses under study. The papers also highlights reported bugs being fixed and the improvement in the compiler's status over a period of time.

Other related efforts to building and using a testsuite include Csmith [17], a comprehensive, well-cited work where the authors perform a randomized test-case generator exposing compiler bugs using differential testing. Such an approach is quite effective to detecting compiler bugs but does not quite serve the validation purpose since it is hard to automatically map a randomly generated failed test to a bug that actually caused it.

LLVM has a testing infrastructure [5] that contains regression tests and whole programs. The tests itself are driven by *lit* testing tool, which is part of LLVM. Recently [3, 4] published examples on how a *user* may program with OpenMP 4.5 on IBM's heterogeneous platforms with GPU support. Though these provide a very good overview on how to use OpenMP 4.5 offload features, they assume that the underlying implementation is as per specification and correct.

3 New in OpenMP 4.x

Going from OpenMP Specification version 3.1 to 4.0, the most significant change was the support for accelerators. OpenMP 4.0 provides directives to describe when data and/or computation should be moved to another computing device/accelerator. This is usually indicated by the presence of `target` keyword which in itself and as a part of others forms the new directives for device offload. Other changes that OpenMP 4.0 brought are: SIMD constructs that enable vectorization of serial and parallelized loops, addition of error handling capabilities, thread affinity mechanisms through OMP_PROC_BIND and OMP_PLACES, tasking extensions for support task-to-task synchronization, Fortran 2003 support that allows interoperability of Fortran and C, user defined reductions, and ability to enforce sequential consistency in atomics [9].

In November 2015 OpenMP Specification 4.5 was released. The significant changes there were the introduction of the `taskloop` construct, allowing the use of `depend, threads and simd` clauses on `ordered` directive, data sharing

DRAFT VERSION

changes allowing C++ methods to privatize accessible non-static members of an object (with restrictions) [10], and changes in default mapping for offloading.

Scalar variables in OpenMP 4.0, in the absence of explicit mapping, were implicitly mapped `tofrom` but in OpenMP 4.5 (without explicit mapping) the scalars are implicitly privatized. They have the same effect as if they were declared `firstprivate` on the target construct. Now the *User* has to explicitly copy the scalar value back if the value is needed on the host. With OpenMP 4.5 C/C++ pointers are implicitly mapped. Hence the host pointer is translated to the corresponding device pointer in case the pointed object is already mapped else it is NULL.

The `use_device_ptr` clause to `target data` construct and `is_device_ptr` clause to `target construct` allow using device specific memory routines. Also new directives such as `target enter data` and `target exit data` were added to enable mapping and un-mapping of variables independently (synchronously or asynchronously) in separate functions or methods. Additional support for mapping C++ references made possible to map structure elements individually in OpenMP 4.5. Asynchronous offloading on the `target` directive through the `nowait` and `depend` clauses is now possible. Lastly the `declare target` directive was extended to be able to mark global variables for deferred mapping.

4 Testsuite Workflow

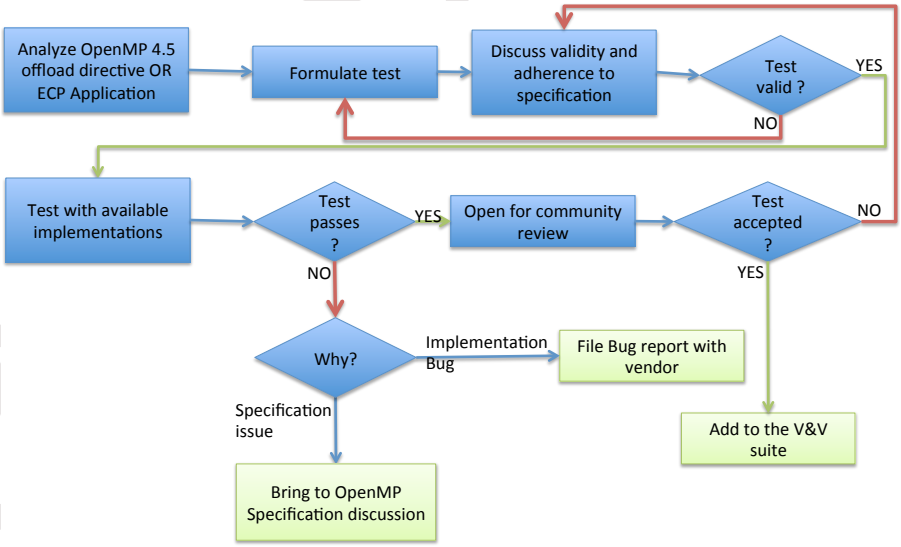

Fig. 1: Workflow for developing the Validation and Verification Suite

The testsuite presented in this work aims at providing tests that provide a comprehensive coverage of different offload directives in OpenMP 4.5. The tests

DRAFT VERSION

are intended to establish accuracy of the interpretation of the OpenMP specification by an implementation and verify the correctness of the functionality. Development of the tests is an on-going iterative process where we address both the functional tests for different combinations of directives and clauses and application kernels tests abstracted out of real world applications. We evaluate all tests by verifying with the specification through a peer review process and testing them on different OpenMP implementations available to us. Our current testbeds (Summitdev, Titan and Summit) at ORNL. They have LLVM, IBMXL, GNU and CCE compilers available, each implementation, in our experience, has different levels of support for OpenMP 4.5. We solicit community feedback, especially from OpenMP developers, to review and refine the tests. For a given directive we first refer to the OpenMP 4.5 specifications to list the different directives and their constructs. Although it is hard to assess how many tests are needed per directive, we try to keep an organized list of the tests we have created. We start by listing all the directives, followed by the multiple constructs that can be used with such directive. For each construct we expand the list with possible modifiers, options, or cases that could apply to that particular construct. Our goal is cover as many valid combinations as possible. We have found it difficult to assess the exact number of tests that are needed per combination of directive and clause(s). In most of the cases, new tests will often come up during the discussion, as there are corner cases either from the description of the specifications, or the interaction with the C and C++ programming model that need to be addressed.

For our second category of tests we collect different application motifs and distill them down to tests through our interaction with different DOE applications. These tests provide insights into how OpenMP 4.5 directives are used in real-world applications and could potentially bring to light unexpected side effects or performance degradation due to interactions between different `target` directives. Currently, our Validation and Verification suite is hosted on bitbucket for easy collaboration and though not complete (in coverage of all offloading directives) we plan to open it to the OpenMP community with this paper. A summary of the results can be seen at our website `https://crpl.cis.udel.edu/ompvvsollve`. The framework, discussed in more detail in Section 5, is geared to be stand-alone, with options to compile for different OpenMP implementations.

Figure 1 represents the development cycle of a test-case. There are three possible positive outcomes of the process we have adopted. Either a test passes through all the checks and makes it to the validation suite, or it uncovers a bug in the vendor implementation of the OpenMP 4.5 standard, or highlights a contentious concept or text that is easy to misinterpret and brings it to the attention of the OpenMP community and specification developers. All tests are written agnostic to where they are executed (host vs. target). We do this to facilitate execution of tests in situations where the host is also configured to be the target or no device is available at the execution time where the computation could be offloaded. After a test executes the output indicates if the test passed or failed and where it was executed (host or target). We have encountered cases

DRAFT VERSION

where we cannot confidently confirm the correctness. One such example is the `nowait` clause. For such cases we do not use affirmative output messages. If the test fails, it could have been because of a number of reasons other than the incorrect implementation and we try to capture it as best as possible. Failure could mean failure to compile or execution time failure that lead to crashes or cryptic error codes. By providing feedback to the vendors we hope to make the error codes more *user* friendly.

5 Validation Suite Infrastructure

In addition to having a well defined workflow to guarantee correctness (as described in Section 4) it is equally important to provide a well defined and flexible infrastructure that supports such a workflow. The design process of this infrastructure has been thoroughly discussed and incrementally improved, resulting in a set of requirements that justify the different features that we have implemented so far. These requirements are as follows:

1. In order to support the previously described workflow, we must define communication mechanisms between active compiler developers, the OpenMP community and the application developers. This requires creation of a web portal and a code repository.
2. It is necessary to provide the *user* with an easy, flexible and simple to use interface. This interface must allow fast deployment and execution of the testsuite. To this end, the testsuite must adapt to new execution environments, providing customization of compilers' options configuration and flags etc. Additionally, it should be able to support different batch schedulers and Linux environment modules.
3. Those who would like to use this testsuite must be able to obtain and export compilation and execution results for an off-line system evaluation. Hence, the designed infrastructure should allow them to obtain results in either a well defined raw logfile format, an intermediate format for exporting to other analysis tools and scripts, or create a well presented report.

With these requirements defined, we present our infrastructure in the rest of this section.

5.1 Development environment and website

As described in our workflow, sharing and evaluating tests is an important part of the test creation process. This requires code sharing and tracking changes. To this end, a repository was created in Bitbucket. We use the features available in a git repository, plus the additional components provided by Bitbucket to support our development workflow. This repository can be found in [14].

In addition to the Bitbucket repository, we have created a website that contains all project related information, including project objectives, documentation on how to use and contribute to this software, publications, and repository guidance. Furthermore, we envision this website as a point of contact

DRAFT VERSION

with those that would use this testsuite, where they can find documentation, examples, and results obtained from the systems evaluated. Our website is `https://crpl.udel.edu/ompvvsollve` .

5.2 Makefile

A Makefile has been created and included in this project. It is used as an entry point to our testsuite. The Makefile allows users to compile, run, and report test results. A set of make rules has been created for each purpose, together with a set of options that modify each rule's behavior (e.g. verbosity, log creation and tests selection). Furthermore, it is possible to use the standard CC and CXX flags to select different compilers. Our testsuite uses the following syntax:

```
make CC=ccompiler CXX=cppcompiler [OPTIONS] [RULE]
```

Rules for Makefile See Table 1 for a list of all the rules that can be used.

Table 1: Set of rules available in the Makefile

Rule	**Description**
compile	Compile tests using the compilers specified by CC and CXX.
run	Run tests that have been previously compiled.
all	Compile and run tests using the compilers specified by CC and CXX
compilers	Print a list of available compilers
report_json	Given a set of logfiles, create a JSON file containing all the results
report_html	Create an HTML-based results report that allows filtering and search.
clean	Remove all compiled tests.

options A set of options can be used to modify the behavior of the rules. The `SOURCES_C` and `SOURCES_CPP` options can be used with `compile` and `all` rules to select which tests to compile. To select what tests to run the `TESTS_TO_RUN` option should be used. The `VERBOSE` option can be used to increase verbosity level of the make command, while `VERBOSE_TESTS` will increase verbosity level of the tests outputs. This is, each test can display additional information at runtime about what it is executing and where the error is encountered. The option `LOG` switches logs on and off, while `LOG_ALL` changes if the output of the Makefile commands should be included in the log files. `NO_OFFLOADING` can be used to turn off offloading capabilities in the compiler.

Other options have been created to adapt the testsuite to the underlying system. It is common to use environment modules to provide multiple software, compilers and libraries within the same system. Additionally, batch schedulers are used to guarantee exclusive and fair access to systems in environments where many users access the same resources. However, this creates new challenges to our

DRAFT VERSION

testsuite. The options MODULE_LOAD and ADD_BATCH_SCHED are available for this purpose. The first one will execute a `module load...` command before compiling or running the tests. The second one will pre-append a batch scheduler command (e.g. jsrun and aprun) to tests that are running. However, since these elements change from system to system it is necessary to create a system description file and use the SYSTEM option to select this description file.

The following use case examples will compile and run all the tests, in verbose mode enabled in both the tests and the make command. Logs will be created including all output from compilers, tests runtime outputs, and make commands outputs. According to the Summit [8] system description file, we will add the `jsrun` command before running each tests, and we will load all the required modules before compiling and running each tests.

```
make CC=clang CXX=clang++ SYSTEM=summit VERBOSE=1 VERBOSE_TESTS=1 \
LOG=1 LOG_ALL=1 ADD_BATCH_SCHED=1 VERBOSE_TESTS=1 MODULE_LOAD=1 all;
```

Customizations As mentioned before, it is possible to customize the testsuite to adapt it to the system environment. A template for a system description file is provided which contains the following options: BATCH_SCHEDULER, C_COMPILER_MODULE, CXX_COMPILER_MODULE, C_VERSION, CXX_VERSION, and CUDA_MODULE. The VERSION commands will be used during the log creation. For further information and example, refer to the documentation on our website.

5.3 Results, Logs and Reports

Although it is possible to obtain results directly from the standard output. We have made it easier for the user to create logs and reports for offline results evaluation. So far there are three options to obtain results: Raw format, JSON format, or HTML format.

Raw format When the LOG option is used in the make command, a new folder called `logs` is created. It contains a log file per test that was compiled and/or executed. Log files are accumulative in the sense that if the make command is issued multiple times, they will all be registered within the same log files. To differentiate multiple runs, as well as compilation from run, we have created a header and footer line per entry, containing system information, compiler used, source file, compiler command, and time. Refer to our website for Header and footer formats.

JSON format Although the raw format is easy to read, it is not well structured and it would be hard to parse and automate to generate final reports. For this reason, we have created the `report_json` rule that uses a script to parse the raw format and output a JSON file. The structure of the JSON file is as follows:

DRAFT VERSION

```
[{
        "Binary path": "...",
        "Compiler command": "...",
        "Compiler ending date": "...",
        "Compiler name": "...",
        "Compiler output": "...",
        "Compiler result": "PASS/FAIL",
        "Compiler starting date": "...",
        "Runtime ending date": "...",
        "Runtime only": false/true,
        "Runtime output": "...",
        "Runtime result": "PASS/FAIL",
        "Runtime starting date": "...",
        "Test comments": "...",
        "Test name": "...",
        "Test path": "...",
        "Test system": "..."
}, ...]
```

HTML format Using the JSON file, it is possible to create a user-friendly and readable report. This report uses a pre-defined HTML template with advanced javascript and css libraries that allows the listing of all the tests in a table, access to more information for each test, filter out results by compiler, systems and PASS/FAIL results or even search an specific name or clause. An snapshot of these results can be seen in Figure 2

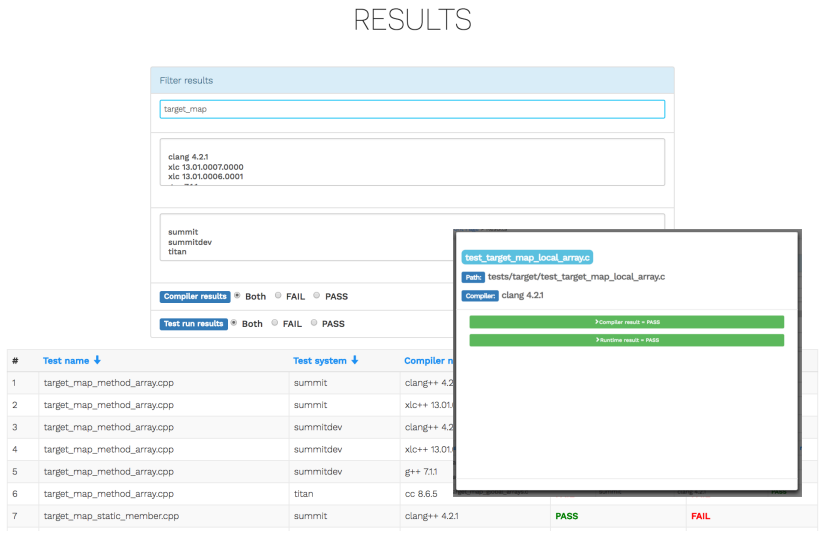

Fig. 2: Snapshot of the HTML report generated by the testsuite

DRAFT VERSION

6 Discussion

Each test is compiled with four different compilers that are available to us. Some of these compilers provide complete support for OpenMP 4.5 constructs while others have indicated to offer only partial support (at the time of writing this paper). By using multiple implementations we are able to analyze the validity of the tests and at the same time how each compiler's implementation behaves for a particular construct under study. The compilers we use include GCC Version 7.1.1, IBM XL Version 14.1 Beta 7, Cray CCE Version 8.6.1 and Clang Version 3.8.0. It is worth noticing that this version of Clang has been modified for our running environment, and it is not exactly the same available in the main LLVM trunk.

We uncovered many implementation bugs, misunderstanding/misinterpretation of the definitions in the specification that led to us (incorrectly) reporting as an implementation bug, as well as ambiguities in the specification throughout this process. Due to space constraints we only discuss the more interesting cases.

6.1 Implementation bugs

- `Target` construct in methods of a class
  During testing `target` directive in C++ methods, we noticed that Clang failed to map an array `tofrom` in one of the OpenMP implementations. Since there is no explicit restriction in the OpenMP specification such usage scenarios are valid. Similarly, a `target` construct in static method of class failed to map class static variables. The later was fixed as a result of the bug report.
- Compiler crashes
  We noticed that the Clang compiler crashed when `collapse` clause was used with dependent iteration spaces. Though the behavior is invalid the implementers agreed that it should present an error to the *user* and not crash. With the Cray implementation, when trying to use `map delete` or `release` of a variable that was originally mapped by `target enter` data, led to compiler crashes. The bug reported was promptly addressed to correct such a behavior.
- Scalar values and `defaultmap`
  For the Cray compiler implementation, we uncovered that the `defaultmap` would not correctly map to the scalar values. Scalar variables in OpenMP 4.0, in the absence of explicit mapping, were implicitly mapped `tofrom` but in OpenMP 4.5 (without explicit mapping) the scalars are implicitly privatized.

From our experience most of the errors on our part were from not accounting for the default mapping on the `target` clause. Earlier in the development phase, especially while trying to test `target data` directive we would run into errors such as an array segment mapping to device with default length (e.g. map(array[1:])) would fail at runtime or we would get errors saying that the test was trying to map data that was already mapped. We understand that this

DRAFT VERSION

behavior is going to change in OpenMP 5.0, where the implicit data mapping on the `target` construct will work differently from explicit data mapping. In the presence of partial mapping, the reference counter will get incremented and it will no longer be classified as undefined behavior.

6.2 Specification Ambiguities

There have been moments when interpreting the specifications document has been problematic. This brought intense discussions in our meetings and/or led to the submission of invalid bug reports. Here we discuss three cases that we would like to go through with the OpenMP community.

When using classes in C++, it is a common practice to use the `this` pointer to refer to the current object, or to access methods or data members of the class. There are difficulties for a programmer using OpenMP to use the `this` pointer for setting data environment as `map(to: this)` or `map(to: this->attribute)` to map the object or an attribute. In the former case, the problem is the interpretation of the `this`. It is interpreted as a pointer but as a special expression. This will cause most of the compilers to complain. In the latter case, the operation `this->attribute` is an arbitrary expression and the `map` clause expects a list item that is mappable.

In the second case, we were attempting to test the `private` and `firstprivate` clauses in combination with offload directives. The specifications document [10] in section 2.15.3.3 says that *"Inside the construct, all references to the original list item are replaced by references to the new list item. In the rest of the region, it is unspecified whether references are to the new list item or the original list item."*. When used with device regions, it is not easy to understand what is the meaning of "rest of the region", as there are three different concepts that must be distinguished: region, target region and task region. It has not been possible for us to clearly identify and separate all these regions, and to understand the rest of the paragraph.

Finally, we present an issue with the array section dependencies. When using the `depend()` clause, it is possible to specify array sections in the form of `depend(inout: a[10:5])`. This clause will map 5 elements of the array `a` starting from position 10. However, in the description of the `depend` clause, section 2.13.9, restriction 1 "List items used in depend clauses of the same task or sibling tasks must indicate identical storage locations or disjoint storage locations.". This implies that it is not possible to map array sections that overlap. Additionally, this section mentions that dependencies only take into consideration storage locations, which creates doubts with respect to the difference between specifying an array section, compared to specifying the element where the array section begins. That is, the difference between `depend(inout: a[10:5])` and `depend(inout: a[10])`. We hope that the specification committee would make requirements more explicit.

DRAFT VERSION

7 Conclusion and Future Work

Our ongoing work on the OpenMP validation and verification test-suite targets features of the OpenMP 4.5 standard in the order of priority as identified by DOE applications. Particularly the offloading mechanisms for *target devices*. Although the majority of our current set of tests are implemented in C and C++, we plan to have Fortran versions in the near future. Our workflow discussed in Section 4 ensures that we make every effort to catch and mitigate implementation or interpretation errors while developing the tests. We try to capture corner cases that we believe might be prone to implementation errors or that are important to applications. Section 5 detailed how to access, execute and customize the testsuite along with how to visualize the results. A website has been built to capture our efforts narrated in this paper `https://crpl.cis.udel.edu/ompvvsollve`. As of this writing we have not covered the entire OpenMP 4.5 API but we hope that this paper will encourage compiler developers and anyone else interested to use the testsuite and provide feedback. While we implement tests for the remainder of the OpenMP 4.5 offloading and new API we want to begin concurrent dialogue with such *users* of the testsuite to ensure smooth adaptability of the V&V suite.

References

1. Clay, M.P., Buaria, D., Yeung, P.K.: Improving scalability and accelerating petascale turbulence simulations using openmp. `http://openmpcon.org/conf2017/program/` (2017), to Appear
2. Friedline, K., Chandrasekaran, S., Lopez, M.G., Hernandez, O.: Openacc 2.5 validation testsuite targeting multiple architectures. In: International Conference on High Performance Computing. pp. 557–575. Springer (2017)
3. Grinberg, L., Bertolli, C., Haque, R.: Hands on with openmp4. 5 and unified memory: Developing applications for ibm hybrid cpu+ gpu systems (part i). In: International Workshop on OpenMP. pp. 3–16. Springer (2017)
4. Grinberg, L., Bertolli, C., Haque, R.: Hands on with openmp4. 5 and unified memory: Developing applications for ibm hybrid cpu+ gpu systems (part ii). In: International Workshop on OpenMP. pp. 3–16. Springer (2017)
5. LLVM: Llvm Testing Infrastructure Guide. `https://llvm.org/docs/TestingGuide.html#test-suite-overview`
6. Müller, M., Neytchev, P.: An openmp validation suite. In: Fifth European Workshop on OpenMP, Aachen University, Germany (2003)
7. Müller, M.S., Niethammer, C., Chapman, B., Wen, Y., Liu, Z.: Validating openmp 2.5 for fortran and c/c++. In: Sixth European Workshop on OpenMP, KTH Royal Institute of Technology, Stockholm, Sweden (2004)
8. Oak Ridge National Laboratory: Summit: Scale new heights. discover new solutions. https://www.olcf.ornl.gov/summit/
9. OpenMP: Openmp - enabling HPC since 1997. `http://www.openmp.org/about/openmp-faq`
10. OpenMP: Openmp 4.5 specification. `http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`

DRAFT VERSION

11. OpenMP: Openmp compilers. http://www.openmp.org/resources/openmp-compilers/
12. Richards, D.F., Bleile, R.C., Brantley, P.S., Dawson, S.A., McKinley, M.S., O?Brien, M.J.: Quicksilver: A proxy app for the monte carlo transport code mercury. In: Cluster Computing (CLUSTER), 2017 IEEE International Conference on. pp. 866–873. IEEE (2017)
13. Top500: Top500 november 2017 list highlights. https://www.top500.org/lists/2017/11/highlights/
14. University of Delaware and Oak Ridge National Laboratory: Openmp 4.5 validation and verification suite. https://bitbucket.org/crpl_cisc/sollve_vv/src
15. Wang, C., Chandrasekaran, S., Chapman, B.: An openmp 3.1 validation testsuite. In: International Workshop on OpenMP. pp. 237–249. Springer (2012)
16. Wang, C., Xu, R., Chandrasekaran, S., Chapman, B., Hernandez, O.: A validation testsuite for OpenACC 1.0. In: Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International. pp. 1407–1416. IEEE (2014)
17. Yang, X., Chen, Y., Eide, E., Regehr, J.: Finding and understanding bugs in c compilers. In: ACM SIGPLAN Notices. vol. 46, pp. 283–294. ACM (2011)

DRAFT VERSION