SOLLVE Verification and Validation OpenMP Testsuite

1st Thomas Huber
University of Delaware
Newark, United States of America
thuber@udel.edu

2nd Swaroop Pophale
Oak Ridge National Laboratory
Oak Ridge, United States of America
pophaless@ornl.gov

3rd Nolan Baker
University of Delaware
Newark, United States of America
nolanb@udel.edu

4th Nikhil Rao
University of Delaware
Newark, United States of America
nikhilr@udel.edu

5th Michael Carr
University of Delaware
Newark, United States of America
mjcarr@udel.edu

6th Jaydon Reap
University of Delaware
Newark, United States of America
jrreap@udel.edu

7th Kristina Holsapple
University of Delaware
Newark, United States of America
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
Oak Ridge, United States of America
lees2@ornl.gov

11th David Bernholdt
Oak Ridge National Laboratory
Oak Ridge, United States of America
bernholdtde@ornl.gov

12th Sunita Chandrasekaran
University of Delaware
Newark, United States of America
schandra@udel.edu

***Abstract*—The SOLLVE V&V suite tests new OpenMP features to visualize compiler & system compliance.**

***Index Terms*—OpenMP, parallelization, testsuite, system compliance**

I. Introduction

Application developers (ADs) often utilize high-performance computing systems to perform lengthy operations in their code. In order to increase program efficiency, these developers frequently use OpenMP. OpenMP is a directive-based programming model which allows for parallelization in C, C++ & Fortran. In the November 2015 release of version 4.5, OpenMP first implemented offloading and utilizing GPUs. Since then, there the following specification releases have been made: 5.0, 5.1, & 5.2 [4]. Despite these advancements, ADs may experience roadblocks when utilizing new OpenMP features. These roadblocks arise if certain features have not been implemented by compiler vendors at the time the developer writes the code. The Scaling OpenMP with LLVM for Exascale performance and portability (SOLLVE) team has created a Verification & Validation (V&V) testsuite to allow ADs to visualize which OpenMP features are currently implemented by various compilers & systems.

In this poster, we illustrate our methodology for test-writing and share an example of a test with a code segment for the OpenMP `masked` directive. Along with this, results from our testsuite are shown for the Oak Ridge National Laboratory (ORNL) Pre-Exascale Systems Summit & Crusher, as well as the National Energy Research Scientific Computing Center's (NERSC) Perlmutter system. Summit features an NVIDIA V100 accelerator, Perlmutter features an NVIDIA A100 accelerator, and Crusher features the AMD accelerator MI250X. These systems allow us to effectively run our suite and test offloading performance with various configurations. The compilers ran on our suite include GNU's GCC, LLVM's Clang, NVIDIA's NVHPC, Cray's CCE, & AMD's ROCm. [3].

Exascale Computing Project

II. Approach

We maintain a consistent methodology to streamline the test-writing process. When the OpenMP specification is updated, any new features are listed in the "B Features History" section [2]. Often, compiler vendors such as LLVM list their implemented features on their websites [1]. The starting point of the test-writing process is comparing OpenMP's features to the features listed per compiler. The new feature is then analyzed, which includes looking at restrictions, arguments, and semantics. Each argument generally requires its own test. A feature may require additional tests if the feature has alternate syntax formats.

Figure 2 shows the workflow behind our project. The majority of tests are written in such a way that the test would fail if a directive was not implemented. In some cases, this

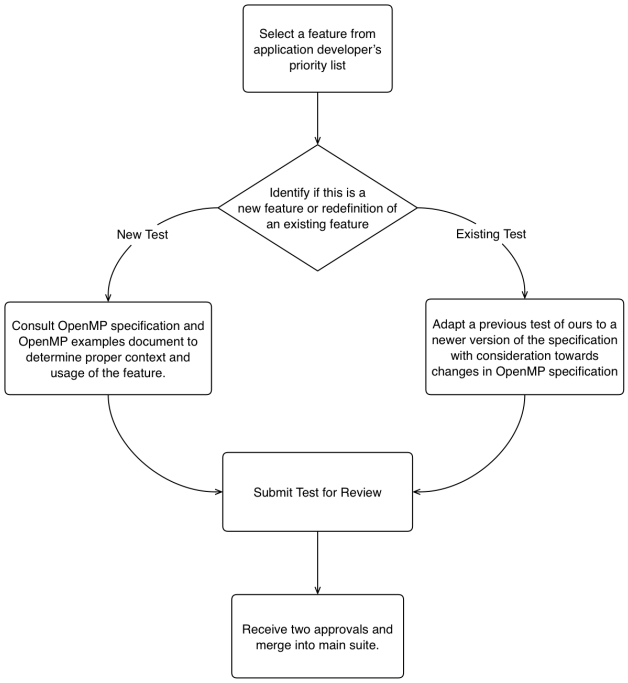

Fig. 1. Process flow demonstrating the typical test creation process.

is not possible. For example, the `assumes` directive changes performance under-the-hood. When this occurs, we ensure that the test does not cause errors when compiling and the rest of the code segment runs as expected. There are three main scenarios in which a test fails:

1) Test was written improperly.
2) Directive has not yet been implemented by compiler of interest.
3) Feature specification was too ambiguous.

In the first scenario, further discussion among the team is required. In the second scenario, the test is run on multiple compilers to visualize other results. In the third scenario, OpenMP specification team or compiler vendors may be contacted. If the test passes, it is then merged into the repository and displayed on the website [3].

III. Results

Results were gathered from three systems: NERSC's Perlmutter system and ORNL's Summit & Crusher systems.

On Perlmutter, GCC 11.2, Clang 15.0.0, NVC 21.11 & CCE 14.0.1 were used. Clang & GCC perform the best with on C tests, with 260 tests passing and 87 failing. LLVM, though, does not have a Fortran compiler available on the system. GCC performs better on other systems which could be due to offloading being disabled by default on Perlmutter. NVIDIA's NVHPC compiler performs the worst on Perlmutter.

On Summit, GCC and Clang both perform relatively better than on Perlmutter. NVHPC has 50 more passing tests compared to NVHPC's performance on Perlmutter. Clang is actually the worst compiler on this system, again, due to its lack of Fortran compiler.

GCC 12.2.1		NVHPC 22.5		LLVM 16.0.0	
Test Name	**Pass/Fail**	**Test Name**	**Pass/Fail**	**Test Name**	**Pass/Fail**
Atomic Compare	Pass	Atomic Compare	Fail	Atomic Compare	Pass
Atomic Compare (device)	Pass	Atomic Compare (device)	Fail	Atomic Compare (device)	Pass
Begin/End Declare Variant	Fail	Begin/End Declare Variant	Fail	Begin/End Declare Variant	Pass
get_mapped_ptr	Pass	get_mapped_ptr	Pass	get_mapped_ptr	Pass
Loop Order Reproducible	Pass	Loop Order Reproducible	Fail	Loop Order Reproducible	Fail
Loop Order Unconstrained	Pass	Loop Order Unconstrained	Fail	Loop Order Unconstrained	Fail
Metadirective (Nothing)	Fail	Metadirective (Nothing)	Fail	Metadirective (Nothing)	Pass
omp_num_teams env variable	Pass	omp_num_teams env variable	Fail	omp_num_teams env variable	Pass
omp_teams_thread_limit env variable	Pass	omp_teams_thread_limit env variable	Fail	omp_teams_thread_limit env variable	Pass
Parallel For Order Reproducible	Pass	Parallel For Order Reproducible	Fail	Parallel For Order Reproducible	Fail
Parallel For Order Unconstrained	Pass	Parallel For Order Unconstrained	Fail	Parallel For Order Unconstrained	Fail
Target Defaultmap(present)	Fail	Target Defaultmap(present)	Fail	Target Defaultmap(present)	Fail
Defaultmap(present) w/ scalar	Fail	Defaultmap(present) w/ scalar	Fail	Defaultmap(present) w/ scalar	Fail
Target has_device_addr	Pass	Target has_device_addr	Fail	Target has_device_addr	Fail
Target memcpy_async_depobj	Pass	Target memcpy_async_depobj	Fail	Target memcpy_async_depobj	Fail
Target memcpy_async_no_obj	Pass	Target memcpy_async_no_obj	Fail	Target memcpy_async_no_obj	Fail
Target memcpy_rect_async_depobj	Pass	Target memcpy_rect_async_depobj	Fail	Target memcpy_rect_async_depobj	Fail
Target memcpy_rect_async_no_obj	Pass	Target memcpy_rect_async_no_obj	Fail	Target memcpy_rect_async_no_obj	Fail
Targetupdate_to_present	Fail	Targetupdate_to_present	Fail	Targetupdate_to_present	Fail
Task nowait	Pass	Task nowait	Fail	Task nowait	Fail
Taskloop Simd Order Reproducible (device)	Pass	Taskloop Simd Order Reproducible (device)	Fail	Taskloop Simd Order Reproducible (device)	Fail
Taskloop Simd Order Unconstrained (device)	Pass	Taskloop Simd Order Unconstrained (device)	Fail	Taskloop Simd Order Unconstrained (device)	Fail

Fig. 2. Table showing GCC 12.2.1, NVHPC 22.5 & LLVM 16.0.0 medium to high priority test results on Summit

On Crusher, the AMD ROCm compiler & Cray CCE compiler perform nearly identical as one another. The compilers on this system have exemplary results, performing better than most other compilers on the other two systems.

Figure 2 shows OpenMP 5.1 test results of features listed as medium-to-high priority by ORNL application teams. This table shows that high priority features have only been implemented by GCC as of yet.

IV. Conclusion

AD teams often need to use OpenMP to increase the efficiency of their code. However, when OpenMP features are not implemented, ensuring AD teams know which systems and compilers perform the best is vital to expediting this process. The SOLLVE OpenMP Testsuite offers open communication between ADs, compiler vendors, OpenMP developers, and other stakeholders. Summit, Perlmutter, and Crusher are all great options for OpenMP with offloading. Our future work is primarily focused on new tests for OpenMP 5.1 and 5.2.

References

[1] Clang 16.0.0 git documentation OpenMP Support. https://clang.llvm.org/docs/OpenMPSupport.html
[2] OpenMP 5.2 Specification. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf#section.B.2
[3] Results :: OpenMP Validation and Verification. https://crpl.cis.udel.edu/ompvvsollve/results/
[4] Specifications - OpenMP. https://www.openmp.org/specifications/