Analysis of Validating and Verifying OpenACC Compilers 3.0 and Above

1st Aaron Jarmusch
University of Delaware
jarmusch@udel.edu

2nd Aaron Liu
University of Delaware
olympus@udel.edu

3rd Christian Munley
University of Delaware
chrismun@udel.edu

4th Daniel Horta
University of Delaware
dchorta@udel.edu

5th Vaidhyanathan Ravichandran
University of Delaware
vaidhy@udel.edu

6th Joel Denny
Oak Ridge National Laboratory
dennyje@ornl.gov

7th Kyle Friedline
University of Delaware
utimatu@udel.edu

8th Sunita Chandrasekaran
University of Delaware
schandra@udel.edu

***Index Terms*—Performance, Programming Model, Testsuite, Validation, Conformance**

I. Overview

Heterogeneous computing creates demand for programming models capable of interfacing and executing on any architectures. OpenACC [1] is a directive-based programming model offering a high level parallelism language handling memory management and functional restrictions based on the hardware architecture. Using *#pragma acc* for C/C++ and sentinel *!$acc* for FORTRAN, the user utilizes a combination of constructs with directives and clauses to parallelize a serial code. The OpenACCV&V testsuite validates and verifies compilers' implementations of OpenACC. This work outlines the compliance of compilers and updates to the testsuite.

II. Design of Tests

Focusing on features that are implementable, we create test utilizing these four steps: interpretation, development, analysis and publication. For example, the *init_if*.c test - here, we initially interpreted that the init directive initializes a compute region. The *if* clause determines whether the region executes on the GPU if true or to execute the region on the host if false. Originally, we created two compute regions with an *if* clause - one that evaluates to true, and one false. Analyzing both the test and the latest OpenACC specification, we found that the directive is not used for a compute region. This realization resulted in a test that initializes the run-time environment based on the *if* clause. We publish tests if they pass or demonstrate missing implementation; this test revealed missing implementation. Sharing the results with compiler developers, it is being implemented for a future update of compilers and completes our workflow until the feature is implemented.

III. Infrastructure

The testsuite contains several hundred tests. To streamline receiving results, an infrastructure is provided to: set up the run-time environment, format results, specify compiler flags, enable or disable conditional compilation, and utilize other optional pre or post processing routines. Made for customization and ease of use, it can be tailored to fit various needs by editing the configuration file. The most important features are: compiler and compiler flags selection, output format specification, conditional compilation, and hang-time limits. Currently, the compilation of C++ tests are not fully supported within the infrastructure.[1]. The primary goal of the LLVM integration is to facilitate the use of the OpenACCV&V Suite for testing the behavior of Clacc's current and future OpenACC support. As a third-party validation testsuite, this assessment of LLVM's (Clacc) conformance to the OpenACC specification can be compared with other implementations.

IV. Results

Utilizing four systems with four versions of various compilers on each, the testsuite tests: NVIDIA's [2] nvc, GNU's GCC [3], HPE/Cray FTN [4], and LLVM's Clacc [5] using the latest and older versions. The tests include features from the latest OpenACC update, not necessarily included in older compiler versions. The results show an increase in the conformance of NVIDIA from nvc 20.9 to nvc 22.5. nvc 22.5 with a Average Pass Rate (APR) of 81.2% across all systems. GNU's GCC also showed improvement from versions 10.1 to 12.1 with the newest GCC compile having an APR of 78.6%. Due to increase in support, HPE/Cray Fortran was tested on Spock with versions 12 and 13 of the compiler. Lastly, Clacc compiler translates OpenACC to OpenMP [6] to utilize the framework developed for Clang by LLVM. A CMakeList file is included

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy 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). This paper is authored by an employee(s) of the United States Government and is in the public domain. Non-exclusive copying or redistribution is allowed, provided that the article citation is given and the authors and agency are clearly identified as its source.

[1]A comprehensive list can be found https://github.com/OpenACCUserGroup/OpenACCV-V

in this project which is the entry point to our llvm-test-suite. It generates the Makefile which allows users to compile, run, and report test results. [2]

V. Analysis of results

The goal of the testsuite is to verify compilers' implementation of OpenACC is in accordance with the specification. Most recently, OpenACC was updated from version 2.5, to 2.7, 3.0, 3.1, and 3.2. Tests for features up to 3.2 are under development with some already within the testsuite. With each compiler implementing these updates individually, the compliance varies between compilers. An observation is that no compiler has a compliance of 100%; the compilers have implemented up to OpenACC 2.6, and are still developing features leading up to 3.2. Some features implemented into the testsuite may not yet be supported by every compiler. This is the major reason why new tests failed; however, as compilers develop they will implement these newer features.

VI. Conclusion

The testsuite's purpose is to validate and verify the implementations outlined within the OpenACC specification for all systems. As seen within the result section, many of the features tested have failed: *serial loop*, *atomic capture*, *routine bind* functions with lambda functions, and the combination of the data clause *copyin* and *copyout* to list a few. Our future goals are to develop tests for new features and more edge cases. We are also going to develop an examples document that will give a beginner friendly breakdown of OpenACC implementation using example code.

VII. Acknowledgements and Reference

We are grateful to Mathew Colgrove, Jeff Larkin, Duncan Poole, Christophe Harle, Guray Ozen, Wael Elwasif, Joel Denny, and Seyong Lee for continued help with this project. OpenACC and NVIDIA have been major supporters in the collaboration and continuation of this project. All names used in this paper are for identification purposes only and may be trademarks respectively.

This research was supported in part through the use of DARWIN computing system: DARWIN – A Resource for Computational and Data-intensive Research at the University of Delaware and in the Delaware Region, Rudolf Eigenmann, Benjamin E. Bagozzi, Arthi Jayaraman, William Totten, and Cathy H. Wu, University of Delaware, 2021, URL: https://udspace.udel.edu/handle/19716/29071

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 used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

[2]LLVM OpenACCV&V Integration can be found here https://github.com/llvm-doe-org/llvm-test-suite/tree/llvm.org/main/External

References

[1] (2022) Openacc: More science, less programming. [Online]. Available: https://www.openacc.org
[2] (2022) High performance computing (hpc) sdk. [Online]. Available: https://developer.nvidia.com/hpc-sdk
[3] (2022) Openacc - gcc wiki. [Online]. Available: https://gcc.gnu.org/wiki/OpenACC
[4] (2022) Cray fortran reference manual (12.0) (s-3901). [Online]. Available: https://support.hpe.com/hpesc/public/docDisplay docId=a00115296en_uspage=OpenACC_Use.html
[5] (2022) Clacc: Openacc support for clang and llvm. [Online]. Available: https://csmd.ornl.gov/project/clacc
[6] J. M. Diaz, S. Pophale, O. Hernandez, D. E. Bernholdt, and S. Chandrasekaran, "Openmp 4.5 validation and verification suite for device offload," in *Evolving OpenMP for Evolving Architectures*, B. R. de Supinski, P. Valero-Lara, X. Martorell, S. Mateo Bellido, and J. Labarta, Eds. Cham: Springer International Publishing, 2018, pp. 82–95.
[7] C. Wang, R. Xu, S. Chandrasekaran, B. Chapman, and O. Hernandez, "A validation testsuite for OpenACC 1.0," in *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, 2014, pp. 1407–1416.
[8] S. Lee, J. Kim, and J. S. Vetter, "OpenACC to fpga: A framework for directive-based high-performance reconfigurable computing," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 544–554.
[9] K. Friedline, S. Chandrasekaran, M. G. Lopez, and O. Hernandez, "Openacc 2.5 validation testsuite targeting multiple architectures," in *High Performance Computing*, J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, Eds. Cham: Springer International Publishing, 2017, pp. 557–575.
[10] M. Wolfe, S. Lee, J. Kim, X. Tian, R. Xu, S. Chandrasekaran, and B. Chapman, "Implementing the openacc data model," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 662–672.