CI/CD Efforts for Validation, Verification and Benchmarking OpenMP Implementations

Aaron Jarmusch[1], Felipe Cabarcas[1,3], Swaroop Pophale[5], Andrew Kallai[1], Johannes Doerfert[2], Luke Peyralans[4], Seyong Lee[5], Joel Denny[5], and Sunita Chandrasekaran[1,6]

[1] University of Delaware, Newark, DE, USA
[2] Lawrence Livermore National Laboratory, Livermore, CA, USA
[3] Universidad de Antioquia, Medellin, Colombia
[4] University of Oregon, Eugene, OR, USA
[5] Oak Ridge National Laboratory, Bethel Valley Road Oak Ridge, TN, USA
[6] {schandra}@udel.edu

Abstract. Software developers must adapt to keep up with the changing capabilities of platforms so that they can utilize the power of High-Performance Computers (HPC), including exascale systems. OpenMP, a directive-based parallel programming model, allows developers to include directives to existing C, C++, or Fortran code to allow node level parallelism without compromising performance. This paper describes our CI/CD efforts to provide easy evaluation of the support of OpenMP across different compilers using existing testsuites and benchmark suites on HPC platforms. Our main contributions include (1) the set of a Continuous Integration (CI) and Continuous Development (CD) workflow that captures bugs and provides faster feedback to compiler developers, (2) an evaluation of OpenMP (offloading) implementations supported by AMD, HPE, GNU, LLVM, and Intel, and (3) evaluation of the quality of compilers across different heterogeneous HPC platforms. With the comprehensive testing through the CI/CD workflow, we aim to provide a comprehensive understanding of the current state of OpenMP (offloading) support in different compilers and heterogeneous platforms consisting of CPUs and GPUs from NVIDIA, AMD, and Intel.

Keywords: OpenMP · Validation · CI · Compiler.· Benchmarking

1 Introduction

Heterogeneous computing has reached a new milestone with the Frontier [21] and Aurora [3] supercomputers achieving exaflops, representing one quintillion floating point operations per second. This significant increase in processing speed has the potential to revolutionize application performance, provided that software developers can keep up with the growing demand for tools and platforms that can harness the power of these devices. To achieve this, current languages and models include OpenMP [24], OpenACC [22], CUDA [20], HIP [5], and OpenCL

[31]. However, each of them has its own unique features and their usage depend on what developers are comfortable with.

While some languages like NVIDIA's CUDA, and AMD's HIP, can be challenging for beginners to learn and require significant rewriting of programs to achieve optimal performance, portable programming models such as OpenMP and OpenACC offer a simpler approach. They use a directive-based approach for parallel programming, allowing users to include these directives on top of existing C, C++, or Fortran code, without compromising on performance. These models play a major role with heterogeneous systems equipped with accelerators such as GPUs, FPGAs, APUs, and more, and can be parallelized on ARM-based systems with ease.

Since more real-world applications have adopted OpenMP, it has been used in various domains. For instance, miniQMC [9], primarily used in the study of electronic molecular structures and 2D/3D solid states, is a simplified version of QMCPACK [14]. Other applications include miniMD [26], LULESH [13], GAMESS [4], HPGMG [7]. With the adoption of OpenMP in modern computing, it is crucial to validate and verify the compilers' implementations of this standard. To simplify the evaluation process for developers, we have established a Continuous Integration (CI) and Continuous Development (CD) workflow, which provides faster feedback for developers. This approach allows us to evaluate the support of OpenMP across different compilers more frequently, reducing the waiting time for developers between evaluations. By implementing this workflow, our goal is to improve the development process and ultimately make OpenMP easier for developers to use in their applications.

This work's focus is on the CI/CD efforts to evaluate the support of OpenMP across different compilers using established validation suites and benchmarks such as OpenMP Validation and Verification (V&V) suite [25], SPEChpc [30], HeCBench [11], and Smoke [1] tests. The main contributions of this paper are:

- Building a Continuous Integration and Continuous Development pipeline for automating testing of OpenMP implementations
- Analysis of output from the CI/CD pipeline that includes verification of OpenMP implementations from AMD, HPE, GNU, LLVM, NVIDIA, and Intel
- Discussion and evaluation of the performance and quality of OpenMP offloading compiler implementations on supercomputers such as Frontier, Perlmutter, Sunspot, and Summit using SPEChpc benchmarking suite.

2 Background & Motivation

OpenMP is a widely used application programming interface (API) that enables programmers to develop parallel applications in C, C++, and Fortran with access to multi-platform shared memory and multi-processing capabilities. With its straightforward adaptable interface, OpenMP enables developers to create efficient parallel programs on multi-core processors. Key features of OpenMP

include directives for parallel programming, runtime library routines, thread management, and environment variables.

Over the years OpenMP has refined support for computations on CPU and, starting with OpenMP 4.0 [23], added support for devices such as accelerators. Although GPGPUs are most widely used accelerator devices used in HPC, OpenMP directives are designed to work with any devices that have memory and are capable of performing computations. That makes OpenMP an attractive choice for future-proofing codes and minimizing developer effort for adapting to different architectural trends.

With multiple vendor implementations of OpenMP and more and more real-world applications porting/developing codes using OpenMP programming model, it is crucial to validate and verify the compilers' implementations of the OpenMP standard. It is often unclear to the application developers if adequate OpenMP features are supported by an implementation and if they are performant. This effort takes that stress away from developers by providing nightly runs of established and curated set of benchmarks and testsuites.

3 Related Work

The current work's focus is to enhance the testing of OpenMP offloading implementations using a CI/CD workflow, which has become increasingly popular in recent years due to its ability to automate various stages of the development process [32,33]. Clacc [8], an open source project that offers support of OpenACC in Clang and LLVM, employs a CI/CD workflow to detect errors quickly. However, the main focus of the current work is to bring a workflow to OpenMP implementation. Even though both OpenACC and OpenMP are directive-based models, their compiler implementations differ, resulting in distinct use cases. LLVM Buildbots[7], another well-known project in the LLVM community, has been used for commit checking of LLVM, but not exclusively for testing OpenMP. However, there are only a few buildbots dedicated to testing OpenMP Offloading, highlighting an area of opportunity for further development. Additionally, CI/CD workflows have also been applied to Machine Learning (ML) through MLOps. Toward MLOps [12], is a case study demonstrating the use of CI/CD workflows to train ML models with different configurations, analyzing the results to identify potential performance bottlenecks and improve the process. Finally, CI/CD workflows have been utilized on the RMACC Summit supercomputer [2] for deploying user-facing software environments and automating benchmark testing [28]. Overall, the current work builds upon these existing efforts to create a more comprehensive and efficient CI/CD workflow for OpenMP Offloading implementations.

[7] https://lab.llvm.org/buildbot/

4 Continuous Integration and Continuous Development Workflow

Manual software testing methods are insufficient for hardware evolution and new software releases. To address this challenge, we use an automation tool to test compilers when they are updated. Leveraging a CI/CD workflow allows us to continuously identify failed tests at all times, not just before an official software release, and to automate the manual testing process. As a result, we reduce the time and effort required for compiler testing, leading to faster development cycles and higher quality software.

4.1 CI/CD Pipeline

We have established an OpenMP CI/CD workflow that involves setup of source code, building and installing the compiler, and suite execution, to reduce the developers time required for testing.

Within our CI/CD workflow a **pipeline** refers to a single run of our workflow, which can be triggered by a scheduled time or by new commits. A pipeline contains **stages** which defines when to run a job. A **job** is a set of steps that defines what you want to accomplish. A pipeline's success depends on the completion of each job, which are labeled as "Pass" or "Fail." If any job within a stage fails, the entire pipeline fails. In other words, each job serves as a building block for the overall pipeline, and its outcome determines whether the pipeline is successful or not. The separation of jobs within a pipeline allows isolation of issues from different parts of an implementation during the collection of source code (setup), build/install of the compiler, or suite execution.

With this idea of separation, as indicated in Figure 1, our workflow (for LLVM Clang/Flang) is divided into three stages: setup, build, testing followed by cleanup.

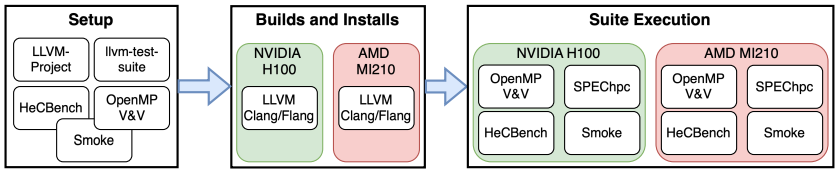

Fig. 1. CI/CD Workflow for LLVM Clang and Flang

In the first stage (i.e., setup), the CI/CD clones any necessary source codes such as the LLVM source, LLVM testsuite infrastructure, and other suites (OpenMP V&V, Smoke, and HeCBench). However, we do not clone SPEChpc suite because the benchmark is not released via GitHub or any other public software storage platform. https://www.spec.org/hpgdownload.html. We download the source codes at a specific commit (for OpenMP V&V, Smoke and HeCBench)

to forgo any new changes coming into the suite after the specific commit we have chosen. For the LLVM compiler source, we download the commit (GitHub commit hash) from the latest trunk.

In the build/install stage, that comes next, we build only the LLVM compiler from the identified commit. We then, proceed to testing with OpenMP V&V, Smoke, HeCBench as well as SPEChpc suite.

In the suite execution stage, the CI/CD is using the LLVM Integrated Tester (LIT) [16] in conjunction with the LLVM-test-suite [17] to compile and execute the OpenMP V&V, HeCBench, and Smoke suites. We designed CMake files for each testsuite, located in the LLVM-test-suite GitHub repository[8]. These CMake files allow us to specify which languages we want to run and which tests or applications to execute. For instance, the OpenMP V&V testsuite consists of a large number of tests, and to make a job pass or fail, we need to provide specific criteria for the CI/CD to determine this. Therefore, we manually ran each suite to create a green and red list of tests for capturing pass and fail results respectively, with each compiler. Additionally, we created green and red lists per accelerator that we target.

In our CI/CD pipelines, we only run the green lists of tests, which ensures reliable and accurate testing results. By integrating both correctness checking and real-world application testing into a single workflow, we can ensure that our compilers are not only correct but also reliable in real-world scenarios.

Experimental Setup: Since hardware evolves at a rapid pace, The University of Oregon (UO) has setup the Frank Cluster which is comprised of a multitude of servers hosted by UO Oregon Advanced Computing Institute for Science and Society (OACISS) with different hardware setups. At the time of writing, across different servers they have an NVIDIA H100, AMD MI210, and an Intel Data Center Max 1100 (Ponte Vecchio). Utilizing the Frank Cluster to implement our first CI/CD to test software means we can easily change to new hardware configurations. For our CI/CD, we are using GitLab and focusing on the AMD MI210 and the NVIDIA H100 within the Frank Cluster. This means we build compilers and execute each suite for both AMD and NVIDIA GPUs for target-specific errors. For instance, we are building LLVM Clang and Flang for AMD and NVIDIA GPUs, which makes two jobs. We also run each suite on both AMD and NVIDIA GPUs. In total, this makes eight jobs. Two and six for the build and testing stages respectively.

On the Frank Cluster we test LLVM Clang and Flang hourly with the OpenMP V&V, HeCBench, and Smoke suites. We run the SPEChpc suite weekly, due to long execution times. The hourly testing is done to provide commit-level feedback to developers because LLVM is open-source and most compiler developers upstream LLVM into their own compiler [15]. All other compilers are tested on a weekly basis with the SPEChpc suite, since the frequency of compiler releases or changes to those compilers are more on a monthly basis. Table 1 is our configuration with the compilers being tested on what hardware. Since we wanted to include testing for Cray we reproduced our CI/CD workflow onto

[8] https://github.com/llvm/llvm-test-suite/tree/main/External

Frontier at ORNL. This also gave us another system to validate our results against. An example of the CI/CD pipeline is shown in Fig. 2.

Table 1. CI/CD pipeline hardware and software configuration.

System		Compilers			
UO Frank Cluster	H100	LLVM Clang & Flang	GNU	NVIDIA	-
UO Frank Cluster	MI210	LLVM Clang & Flang	GNU	AMD	-
ORNL Frontier	MI250X	LLVM Clang & Flang	GNU	AMD	Cray

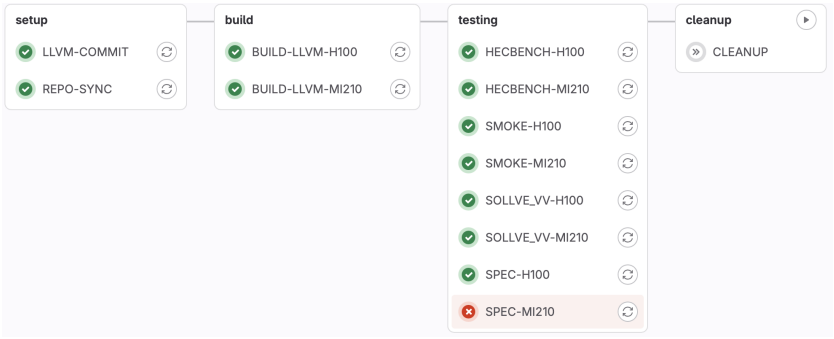

Fig. 2. The CI/CD output for a LLVM Clang and Flang pipeline. The green checkmark marks the job as "Pass" while the red 'x' marked the job as "Fail".

Discussion: The development and testing of compilers require rigorous testing to ensure quality. While automating these tasks can accelerate the process, it's crucial to maintain effective communication with developers to provide timely feedback on testsuites' failures. To address this challenge, we have implemented two forms of automatic communication. First, for LLVM, we have integrated a messaging service to notify a team of compiler developers interested in LLVM's implementation of OpenMP offloading when a pipeline has failed. This provides the developers with immediate feedback in case a recent commit has broken the compiler. However, due to LLVM's open-source nature and large project aspect, it may not be possible to identify the root cause of a build failure solely through automated means, it may or may not be the fault of LLVM Clang or Flang OpenMP offloading implementation. The cause may have risen from other parts of the LLVM toolchain. Therefore, we still require some form of human interference to differentiate between failed jobs and identify the specific issue.

Similarly, build-bots have been used in LLVM to test every commit. These build-bots notify the commit creator if a failure occurs; however they primarily focus on CPU code testing and building. Consequently, our primary concern is not build failures but failures within the testsuites themselves. Therefore, it

might not be advantageous to notify commit creators of these failures. Instead, we should track them specifically for developers working on OpenMP Offloading.

For the second form of communication, we are utilizing the LLVM Nightly Testing (LNT) [18] infrastructure to create a server that stores testsuite information about the kernel. This information is visualized in a graph to track performance over time[9]. Overall, these methods of communication with developers about the results of the pipelines have been effective so far. However, we continue to evaluate and improve our approaches to ensure optimal effectiveness in the development and testing of compilers.

5 OpenMP Validation & Verification Suite and SPEChpc Benchmarking Suite

In this section, we will elaborate further on the OpenMP V&V testsuite and SPEChpc benchmarking suite exercised by our CI/CD workflow. The OpenMP V&V testsuite is primarily concerned with verifying compiler correctness, whereas the SPEChpc benchmarking suite evaluates the quality of compiler implementations. By integrating both correctness verification and application-based benchmarking into a CI/CD workflow, we can ensure more robust and reliable compiler testing.

5.1 OpenMP Validation and Verification Suite

The OpenMP Validation and Verification (OpenMP V&V) suite [25,10] started as a sub-project of the SOLLVE (Scaling OpenMP With LLVM for Exascale) Exascale Computing Project (ECP) [29]. The sub-project was born out of the lack of open, unbiased testsuite to evaluate aspects of OpenMP that were most critical for exascale applications. Yet, there is still more work to be done. ECP marked a significant achievement with the push towards exascale computing resources. ECP ended in December 2023. As a path forward, the Next Generation Science Software (S4PST) project [27] aims to push the boundaries even further. This focuses on creating a more predictive ecosystem for sustainability in HPC software, with a particular emphasis on node-level programming systems and tools. The tests in the OpenMP V&V are organized based on the specification version. They contain feature and kernel tests in C, C++ and Fortran.

Coverage: Table 2 shows there are 742 total tests in the testsuite covering new features starting with the OpenMP 4.5 specification up to the 5.2. The OpenMP testsuite is under active development and the numbers here represent a snapshot at the time of this writing. The number of test per specification is mainly related to the number of new features introduced in the specification.

Results: Figure 3 shows the number of tests that pass for each compiler in each system, for C/C++ and Fortran separately[10]. The left axis represents

[9] https://crpl.cis.udel.edu/lnt-sollve/

[10] `zenodo.org/doi/10.5281/zenodo.12571032`: allCompilerSystemsResults.json

Table 2. Number of tests in C, C++, and Fortran for each OpenMP Specification currently in the OpenMP Validation and Verification Suite.

Version	C	C++	Fortran	Total
4.5	134	14	104	252
5.0	191	13	128	332
5.1	99	2	28	129
5.2	16	8	5	29
Total	440	37	265	742

Table 3. Systems and compiler versions used for validation

System	Vendor	Accelerator	Compiler and Versions
Perlmutter	HPE	NVIDIA A100	NVIDIA 24.5, Cray 17.0.0, LLVM 19.0.0 commit (18ec885a), and GNU 14.1
Frontier	HPE	AMD MI250X	AMD's ROCm 6.0.0, Cray 17.0.0, LLVM 19.0.0 commit (18ec885a), and GNU 14.1
Sunspot	Intel	Intel GPU Max Series	OneAPI 18.0.0

C/C++, while the right is Fortran. AMD C/C++ compiler on Frontier and GNU on Perlmutter pass the most, with GNU is the compiler that passes most tests over all. In the case of Fortran, in both systems GNU passes the most tests. LLVM Flang for offloading is under development. Table 4 shows the results for each OpenMP Version.

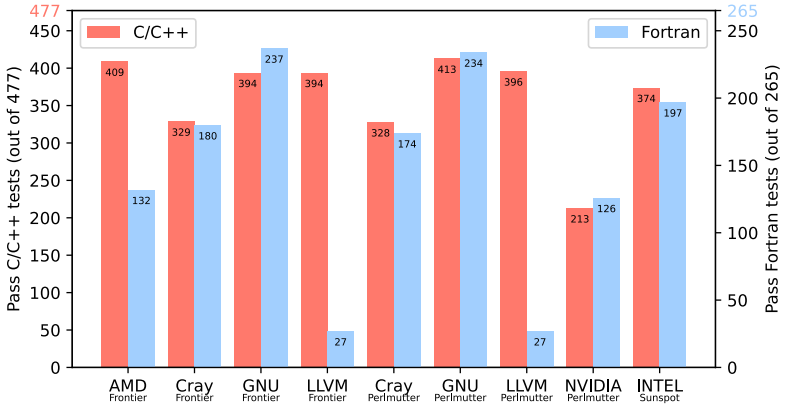

Fig. 3. No. of tests passing per system out of total 477 C/C++ & 265 Fortran tests

Figure 4 shows the compiler evolution over a period of two years for Perlmutter and Frontier[11]. The chosen compilers are the recommended compilers in the given system for OpenMP offloading application development. It can be seen in

[11] `zenodo.org/doi/10.5281/zenodo.12571032`: compilerVersionsPerlmutterFrontierResults.json

Table 4. OpenMP Validation and Verification Suite pass results per OpenMP Version.

		Frontier				Perlmutter				Sunspot
Ver.	**Lang.**	**AMD**	**Cray**	**GNU**	**LLVM**	**Cray**	**GNU**	**LLVM**	**NVIDIA**	**INTEL**
4.5	C & C++	146	142	137	147	142	145	148	131	142
5.0		179	147	170	171	146	175	172	67	162
5.1		68	39	75	66	39	75	66	13	67
5.2		16	1	12	10	1	18	10	2	3
Total		**409**	**329**	**394**	**394**	**328**	**413**	**396**	**213**	**374**
4.5	Fortran	86	89	104	15	88	104	15	97	97
5.0		40	86	110	9	81	107	9	24	85
5.1		2	3	19	0	3	19	0	2	12
5.2		4	2	4	3	2	4	3	3	3
Total		**132**	**180**	**237**	**27**	**174**	**234**	**27**	**126**	**197**

the graph that the compiler development for Fortran has been slow during this period. In the case of C/C++ there has been greater improvement over time, specially for LLVM. NVIDIA supports till OpenMP 4.5.

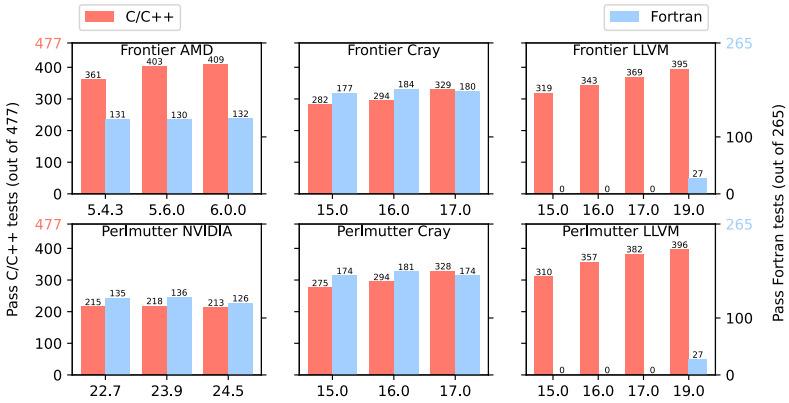

Fig. 4. Evolution of compilers on Frontier and Perlmutter

5.2 SPEChpc HPG Benchmarking Suite

We present SPEChpc V1.1.8 HPG [30] benchmark for evaluating the performance and quality of OpenMP offloading features [6]. The benchmark contains 9 MPI applications written in different programming models, including OpenMP-offloading (TGT) and OpenACC (ACC), where 6 of them are C/C++ and 3 are Fortran. While we focus on the OpenMP offloading versions of the applications, we also present the OpenACC results as a performance reference of another

directive programming model. Table 5 summarizes the SPEChpc applications. We use the smallest size (tiny) versions of the benchmarks, which can run on a single node of each of the tested systems. In addition to these tests, we include a modified *532.sph_exa*, which we will reference as *532.sph_exaM*. The modified version was a result of code analysis as we noticed a bigger performance difference with the OpenACC version compared to OpenACC (we discuss this in more detail at the latter half of this section).

Table 5. SPEChpc 2021 version 1.1.8 applications taken from SPEC website [30].

Benchmark Name	Size: Tiny	Language	Application Area
LBM D2Q37	505.lbm	C	Computational Fluid Dynamics
SOMA	513.soma	C	Polymeric Systems
Tealeaf	518.tealeaf	C	High Energy Physics
Cloverleaf	519.clvleaf	Fortran	High Energy Physics
Minisweep	521.miniswp	C	Nuclear Engineering
POT3D	528.pot3d	Fortran	Solar Physics
SPH-EXA	532.sph_exa	C++14	Astrophysics and Cosmology
HPGMG-FV	534.hpgmgfv	C	Cosmology, Astrophysics
miniWeather	535.weather	Fortran	Weather

SPEChpc Results: Table 6 presents estimated base time results of SPEChpc 2021 Version 1.1.8 on Frontier[12] and Perlmutter[13] using available compilers.

Table 6. Estimated **Base Run Time** of SPEChpc tiny applications on Perlmutter and Frontier using one node. **EE** is execution error, while **BE** is build error.

	Estimate Base Time (seconds)							
	Perlmutter				**Frontier**			
Compiler	**GNU**	**LLVM**	**Cray**	**NVIDIA**	**GNU**	**LLVM**	**Cray**	**ROCm**
Version	**14.1**	**19.0.0**	**17.0.0**	**24.5**	**14.1**	**19.0.0**	**17.0.0**	**6.0.0**
505.lbm	484.89	38.29	*28.34*	35.9	2813.46	43.44	40.82	54.64
513.soma	855.05	69.61	*56.75*	65.64	**BE**	87.98	**BE**	70.05
518.tealeaf	2200.95	90.84	49.09	40.49	337.12	43.58	*40.41*	48.51
519.clvleaf	**BE**	**BE**	**EE**	*45.54*	**BE**	**BE**	58.73	72.73
521.miniswp	**EE**	209.55	96.76	573.09	**EE**	160.44	*93.59*	142.61
528.pot3d	926.24	**BE**	55.34	61.54	**BE**	**BE**	*45.64*	92.61
532.sph_exa	1454.46	849.6	**EE**	491.41	**BE**	*203.34*	226.33	207.4
532.sph_exaM	5973.46	179.41	*128.36*	**EE**	**BE**	145.61	164.87	144.83
534.hpgmgfv	**EE**	156.75	*71.2*	163.33	**BE**	102.32	87.5	95.59
535.weather	1391.84	**BE**	38.51	42.72	2569.96	**BE**	*32.51*	53.19

[12] `zenodo.org/doi/10.5281/zenodo.12571032`: Frontier_SPEC_PaperResults.zip

Looking at the results from the perspective of the applications, only **505.lbm** and **518.tealeaf** can be built and executed with all compilers (both are C/C++ applications). Fortran's **519.clvleaf** can only be built and executed by NVIDIA's compiler in Perlmutter and by AMD's ROCm and Cray compilers in Frontier. From the point of view of the compilers, only AMD's ROCm compiler could build and execute all applications. The performance of LLVM C/C++ applications is similar to that from vendor compilers. LLVM Flang is under development so it could not build Fortran applications. Even though the GNU compiler can build Fortran and C/C++ applications, its performance is very low compared to all others, specially on Frontier, where many of the applications don't even build. This is an interesting observation, as we found GNU compilers doing a better job with correctness (Figure 3).

As mentioned before, AMD's compiler was the only one that built and executed all applications with good performance. For this to happen, AMD required special flags suggested by the developers, like (*-fopenmp-target-xteam-reduction-blocksize=128*, *-Mx,201,2*, *-fno-openmp-assume-no-nested-parallelism*, *–mca topo basic*). Finally, as a performance reference for the Perlmutter system, we present SPEChpc results for the OpenACC programming model[13] as shown in Table 7. We noticed significant performance differences between OpenMP and OpenACC outputs for some applications, especially **532.sph_exa** and **521.miniswp**. As mentioned before, a direct comparison of results from both the models should not and cannot be made without understanding the intricate details of feature implementations and understanding the reasoning behind performance gaps.

Listing 1.1 contains the OpenMP annotation added to **532.sph_exa**. We found that this memory allocation on the device was missing for the OpenMP offloading version, while it was present for the OpenACC version. The modified version is referenced in this document as **532.sph_exaM**[14]. In the original OpenMP version, because these pragmas were missing, every time these arrays were accessed in a target region they have to be allocated in the device. By adding these lines, the arrays are allocated in the device before multiple iterations of device kernel calls. In Perlmutter, while the original version does not execute correctly with Cray compiler (it is a memory allocation error), the modified version doesn't run correctly with NVIDIA and GNU Compilers (it is also a memory allocation error). In the case of Frontier, neither versions compile with GNU, but the error doesn't have anything to do with the modification, it is a compiler bug.

Listing 1.1. These are the pragmas added to the benchmark **532.sph_exa**. The new benchmark is **532.sph_exaM**. These lines were added to the original file **SqPatch.hpp** on the function **resizeN(size_t size)**.

```
// OpenACC annotation present in the original code
#pragma acc exit data delete(hNptr, hNCptr)
// First OpenMP Directive added
#pragma omp target exit data map(delete: hNptr[:Nsze],
```

[13] zenodo.org/doi/10.5281/zenodo.12571032: Perlmutter_SPEC_PaperResults.zip
[14] zenodo.org/doi/10.5281/zenodo.12571032: SqPatch.hpp.diff

```
                                                hNCptr[:NCsze])

// OpenACC annotation present in the original code
#pragma acc enter data create(hNptr[:size*ngmax],
                              hNCptr[:size])
// Second OpenMP Directive added
#pragma omp target enter data map(alloc: hNptr[:size*ngmax],
                                         hNCptr[:size])
```

Table 7. OpenACC estimated **base run time** of SPEChpc tiny applications on Perlmutter with NVIDIA compiler.

	Estimate Base Time (seconds)								
Version	**505 lbm**	**513 soma**	**518 tealeaf**	**519 clvleaf**	**521 miniswp**	**528 pot3d**	**532 sph_exa**	**534 hpgmgfv**	**535 weather**
24.5	28.48	45.82	48.23	35.69	52.38	53.58	129.08	64.27	37.23

Discussion: We observed that it should not be concluded that one compiler is better over the other as the results largely depends on the applications and optimizations. Having said that, Cray demonstrated some of the best results. Furthermore, Cray's results on Perlmutter are very similar to the OpenACC results for **505.lbm**, **528.pot3d** and **535.weather**. Moreover, the results of **532.sph_exaM** for Cray compiler is similar to the OpenACC version of **532.sph_exa**. These also suggest that the large performance gaps for codes such as **521.miniswp** between OpenMP and OpenACC could be an implementation difference, not necessarily a compiler optimization difference. SPEChpc uses only 4.5 specification features, but as seen, the compilers are still having trouble, after almost 10 years of the release of the specification, to produce efficient code for all applications. Moreover, the performance is not as good as the OpenACC versions. As seen with **532.sph_exa**, the lack of performance of the applications may not be caused only by compiler optimization issues, but it could also be application implementation differences as a result of not using OpenMP features that compilers don't support or that their performance is low. The performance of larger versions of SPEChpc would correlate to the results shown here, but they would also depend on the network, and the performance of the MPI implementation. We expect that the performance per node would be similar, but we can not extrapolate to the performance of larger versions of the benchmark.

6 Conclusion

This paper sheds light on the pass/fail of OpenMP features from version 4.5 and above. We do so by building a CI/CD workflow comprising of OpenMP V&V, Smoke, HeCBench, and SPEChpc suites to automate the testing of OpenMP

implementations with a goal to fail fast and provide feedback to LLVM compiler developers especially since several vendor compilers are LLVM-based. This paper further provides details of the current status of correctness of OpenMP compilers via the V&V testing infrastructure and tests running on Frontier, Perlmutter, and Sunspot. As we know, manual test creation is a time consuming process for developers; this challenge is currently being explored by using Large Language Models (LLM) for test generation [19]. The work also discusses the quality of OpenMP compiler implementations on different GPUs when using SPEChpc benchmarking suite. The discussion also offers suggestions and room for further implementation improvement.

Building on the workflow developed in this work, future works could include the implementation of "expected fail" testing, the continous development of tests as the specification evolves, and enabling developers to reproduce and debug problems effectively.

In conclusion, the OpenMP offloading features represent a crucial environment for the HPC community as the systems continue to evolve. The model has been bringing legacy and new applications to run on novel systems, which makes it time critical to closely track correctness and quality of implementations guiding the developers accordingly.

Acknowledgments This research used resources of the OLCF at ORNL supported by the Office of Science of the U.S. DOE under Contract No. DE-AC05-00OR22725; used resources of the ALCF, a U.S. DOE Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-ASCR program, under Contract No. DE-AC02-06CH11357; used resources of NERSC, a U.S. DOE Office of Science User Facility located at LBNL, operated under Contract No. DE-AC02-05CH11231 using NERSC ERCAP0029463. This material is also based upon work supported by the U.S. DOE under Contract DE-FOA-0003177, S4PST: Next Generation Science Software Technologies Project.

References

1. AMD: AMD ROCm Smoke Tests, `https://github.com/ROCm/aomp/tree/aomp-dev/test/smoke`
2. Anderson, J., Burns, P., Milroy, D., Ruprecht, P., Hauser, T., Siegel, H.: Deploying rmacc summit: An hpc resource for the rocky mountain region. pp. 1–7 (07 2017). `https://doi.org/10.1145/3093338.3093379`
3. Argonne National Laboratory: Argonne's aurora supercomputer breaks exascale barrier (May 2024), `https://www.anl.gov/article/argonnes-aurora-supercomputer-breaks-exascale-barrier`
4. Bak, S., Bertoni, C., Boehm, S., Budiardja, R., Chapman, B.M., Doerfert, J., Eisenbach, M., Finkel, H., Hernandez, O., Huber, J., et al.: Openmp application experiences: Porting to accelerated nodes. Parallel Computing **109**, 102856 (2022)
5. Bauman, P., Chalmers, N., Curtis, N., Freitag, C., Greathouse, J., Malaya, N., McDougall, D., Moe, S., van Oostrum, R., Wolfe, N., et al.: Introduction to amd gpu programming with hip. Presentation at Oak Ridge National Laboratory. Online

at: https://www. olcf. ornl. gov/calendar/intro-to-amd-gpu-programming-with-hip (2019)
6. Brunst, H., Chandrasekaran, S., Ciorba, F.M., Hagerty, N., Henschel, R., Juckeland, G., Li, J., Vergara, V., Wienke, S., Zavala, M.: First experiences in performance benchmarking with the new spechpc 2021 suites. In: 2022 22nd International Symposium on Cluster, Cloud and Internet Computing (CCGrid). pp. 675–684. IEEE Computer Society, Los Alamitos, CA, USA (may 2022). https://doi.org/10.1109/CCGrid54584.2022.00077, https://doi.ieeecomputersociety.org/10.1109/CCGrid54584.2022.00077
7. Daley, C., Ahmed, H., Williams, S., Wright, N.: A case study of porting hpgmg from cuda to openmp target offload. In: OpenMP: Portable Multi-Level Parallelism on Modern Systems: 16th International Workshop on OpenMP, IWOMP 2020, Austin, TX, USA, September 22–24, 2020, Proceedings 16. pp. 37–51. Springer (2020)
8. Denny, J.E., Lee, S., Vetter, J.S.: Clacc: Translating openacc to openmp in clang. In: 2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC). pp. 18–29. IEEE (2018)
9. Huber, J., Cornelius, M., Georgakoudis, G., Tian, S., Diaz, J.M.M., Dinel, K., Chapman, B., Doerfert, J.: Efficient execution of openmp on gpus. In: 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). pp. 41–52. IEEE (2022)
10. Huber, T., Pophale, S., Baker, N., Carr, M., Rao, N., Reap, J., Holsapple, K., Davis, J.H., Burnus, T., Lee, S., Bernholdt, D.E., Chandrasekaran, S.: Ecp sollve: Validation and verification testsuite status update and compiler insight for openmp. In: 2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC). pp. 123–135 (2022). https://doi.org/10.1109/P3HPC56579.2022.00017
11. Jin, Z., Vetter, J.S.: A benchmark suite for improving performance portability of the sycl programming model. In: 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 325–327 (2023). https://doi.org/10.1109/ISPASS57527.2023.00041
12. John, M.M., Olsson, H.H., Bosch, J.: Towards mlops: A framework and maturity model. In: 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 1–8 (2021). https://doi.org/10.1109/SEAA53835.2021.00050
13. Karlin, I., Scogland, T., Jacob, A.C., Antao, S.F., Bercea, G.T., Bertolli, C., de Supinski, B.R., Draeger, E.W., Eichenberger, A.E., Glosli, J., et al.: Early experiences porting three applications to openmp 4.5. In: OpenMP: Memory, Devices, and Tasks: 12th International Workshop on OpenMP, IWOMP 2016, Nara, Japan, October 5-7, 2016, Proceedings 12. pp. 281–292. Springer (2016)
14. Kim, J., Baczewski, A.D., Beaudet, T.D., Benali, A., Bennett, M.C., Berrill, M.A., Blunt, N.S., Borda, E.J.L., Casula, M., Ceperley, D.M., et al.: Qmcpack: an open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids. Journal of Physics: Condensed Matter **30**(19), 195901 (2018)
15. Lambert, J., Monil, M.A.H., Lee, S., Malony, A.D., Vetter, J.S.: Leveraging compiler-based translation to evaluate a diversity of exascale platforms. In: 2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC). pp. 14–25 (2022). https://doi.org/10.1109/P3HPC56579.2022.00007
16. LLVM Compiler Infrastructure: lit - LLVM Integrated Tester, https://llvm.org/docs/CommandGuide/lit.html

17. LLVM Compiler Infrastructure: LLVM Test Suite, https://github.com/llvm/llvm-test-suite
18. LLVM Compiler Infrastructure: LNT infrastructure for performance testing, https://llvm.org/docs/lnt/tests.html
19. Munley, C., Jarmusch, A., Chandrasekaran, S.: Llm4vv: Developing llm-driven testsuite for compiler validation. Future Generation Computer Systems **160**, 1–13 (2024). https://doi.org/https://doi.org/10.1016/j.future.2024.05.034, https://www.sciencedirect.com/science/article/pii/S0167739X24002449
20. NVIDIA: CUDA, https://developer.nvidia.com/cuda-toolkit
21. Oak Ridge National Laboratory: Frontier supercomputer debuts as world's fastest, breaking exascale barrier (May 2022), https://www.ornl.gov/news/frontier-supercomputer-debuts-worlds-fastest-breaking-exascale-barrier
22. OpenACC Organization: OpenACC, https://www.openacc.org/
23. OpenMP Architecture Review Board: OpenMP application program interface version 4.0 (2013), https://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf
24. OpenMP Architecture Review Board: OpenMP application program interface version 5.2 (2021), https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf
25. ORNL and University of Delaware: OpenMP validation and verification suite, https://github.com/OpenMP-Validation-and-Verification/OpenMP_VV
26. Pennycook, S.J., Sewall, J.D., Hammond, J.R.: 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). pp. 37–46. IEEE (2018)
27. S4PST: Sustainability for Programming Systems and Tools: S4PST, https://ornl.github.io/events/s4pst2023/
28. Sampedro, Z., Holt, A., Hauser, T.: Continuous integration and delivery for hpc: Using singularity and jenkins. In: Proceedings of the Practice and Experience on Advanced Research Computing. PEARC '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3219104.3219147, https://doi.org/10.1145/3219104.3219147
29. SOLLVE: Scaling OpenMP With LLVm for Exascale Performance and Portability : SOLLVE, https://www.exascaleproject.org/research-project/sollve/
30. Standard Performance Evaluation Corporation: SPEChpc™ 2021 benchmark suites, https://www.spec.org/hpc2021/
31. The Khronos Group: OpenCL, https://www.khronos.org/opencl/
32. Varrette, S., Bouvry, P., Cartiaux, H., Georgatos, F.: Management of an academic hpc cluster: The ul experience (07 2014). https://doi.org/10.1109/HPCSim.2014.6903792
33. Yu, L., Alégroth, E., Chatzipetrou, P., Gorschek, T.: A roadmap for using continuous integration environments. Commun. ACM **67**(6), 82–90 (may 2024). https://doi.org/10.1145/3631519, https://doi.org/10.1145/3631519