ECP SOLLVE Validation and Verification OpenMP Offloading Testsuite

Thomas Huber[1], Swaroop Pophale[3], Nolan Baker[1], Nikhil Rao[1], Michael Carr[1], Jaydon Reap[1], Kristina Holsapple[1], Joshua Hoke Davis[4], Tobias Burnus[5], Seyong Lee[3], David E. Bernholdt[3], Sunita Chandrasekaran[1, 2]

[1]University of Delaware, [2] Brookhaven National Laboratory, [3]Oak Ridge National Laboratory, [4]University of Maryland, [5]Siemens Electronic Design Automation

ABSTRACT

Scaling **O**penMP with **LL**VM for **E**xascale performance and portability (**SOLLVE**) aims to scale OpenMP by leveraging LLVM for exascale performance and portability of applications. A Verification & Validation (V&V) testsuite tests various OpenMP directives to evaluate system & vendor compliance. The V&V suite is ran on various systems, including NERSC's Perlmutter system & Oak Ridge National Laboratory's Summit & Crusher systems.

BACKGROUND INFO

OpenMP, a parallel-based programming model, allows for performance optimization in C, C++ & Fortran, with its features called "directives" listed in the OpenMP specification (spec). The purposes of the SOLLVE Verification and Validation (V&V) testsuite are to:

- Evaluate compiler's compliance with the specification
- Identify ambiguities in the specification
- Illustrate a system's ability to run OpenMP directives & utilize offloading parallel directives on GPUs
- Demonstrate the use & purpose of new OpenMP directives to application developers

OpenMP is useful for many application developers working on HPC systems to ensure their code is running at maximum efficiency. Our testsuite ensures that the OpenMP specification, compiler vendors & system operators are implementing OpenMP effectively.

APPROACH

- Tests are usually written to ensure the test would fail if the directive were to not work properly.
- If the test **fails**, analysis is done to determine the issue:
  - The test was written improperly
  - The directive has not yet been implemented
  - The specification is too ambiguous
- If the test **passes**, it is merged into the repository.
- Results for each system & compiler are then uploaded to the website.

TEST EXAMPLE

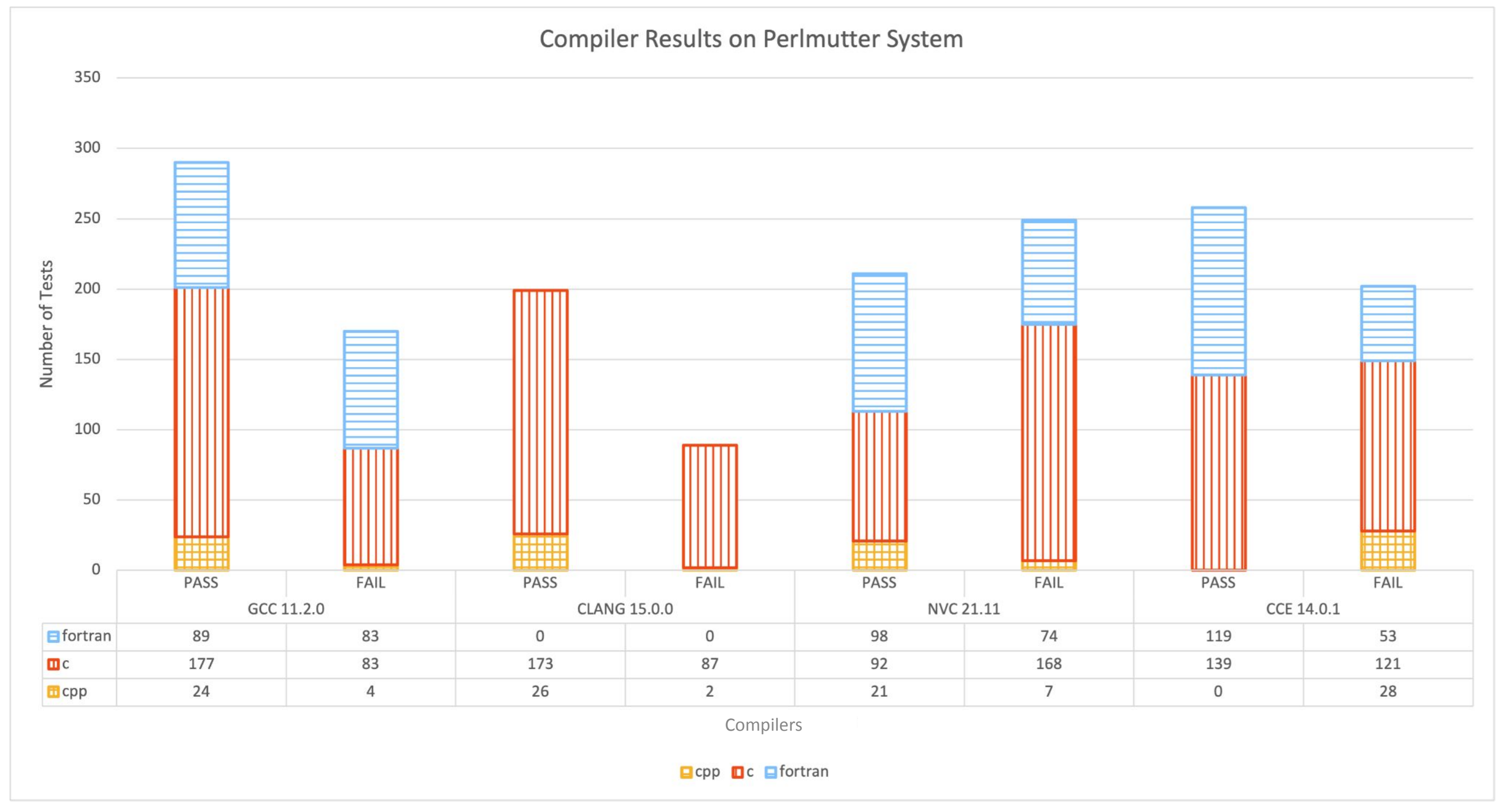
```
#pragma omp parallel num_threads(threads)
while(1){
 int tot;
 #pragma omp atomic read
 tot = total;
 if (tot <= 0)
    break;
  #pragma omp masked
  {
    OMPVV_TEST_AND_SET_VERBOSE(errors, omp_get_thread_num() !=
0); // primary thread
    ct++;
    #pragma omp atomic
    total = total-1;
  }
}
 OMPVV_TEST_AND_SET_VERBOSE(errors, ct != 10);
```

Figure 1: test_masked.c code segment

This test, which focuses on the **omp masked** directive which only runs on the master thread, demonstrates:

- `OMPVV_TEST_AND_SET_VERBOSE` function, which is part of the V&V suite and is a primary tool in reporting errors for test results
- Errors reported if the running thread is not the master (first) thread
- Errors reported if masked code segment does not run 10 times

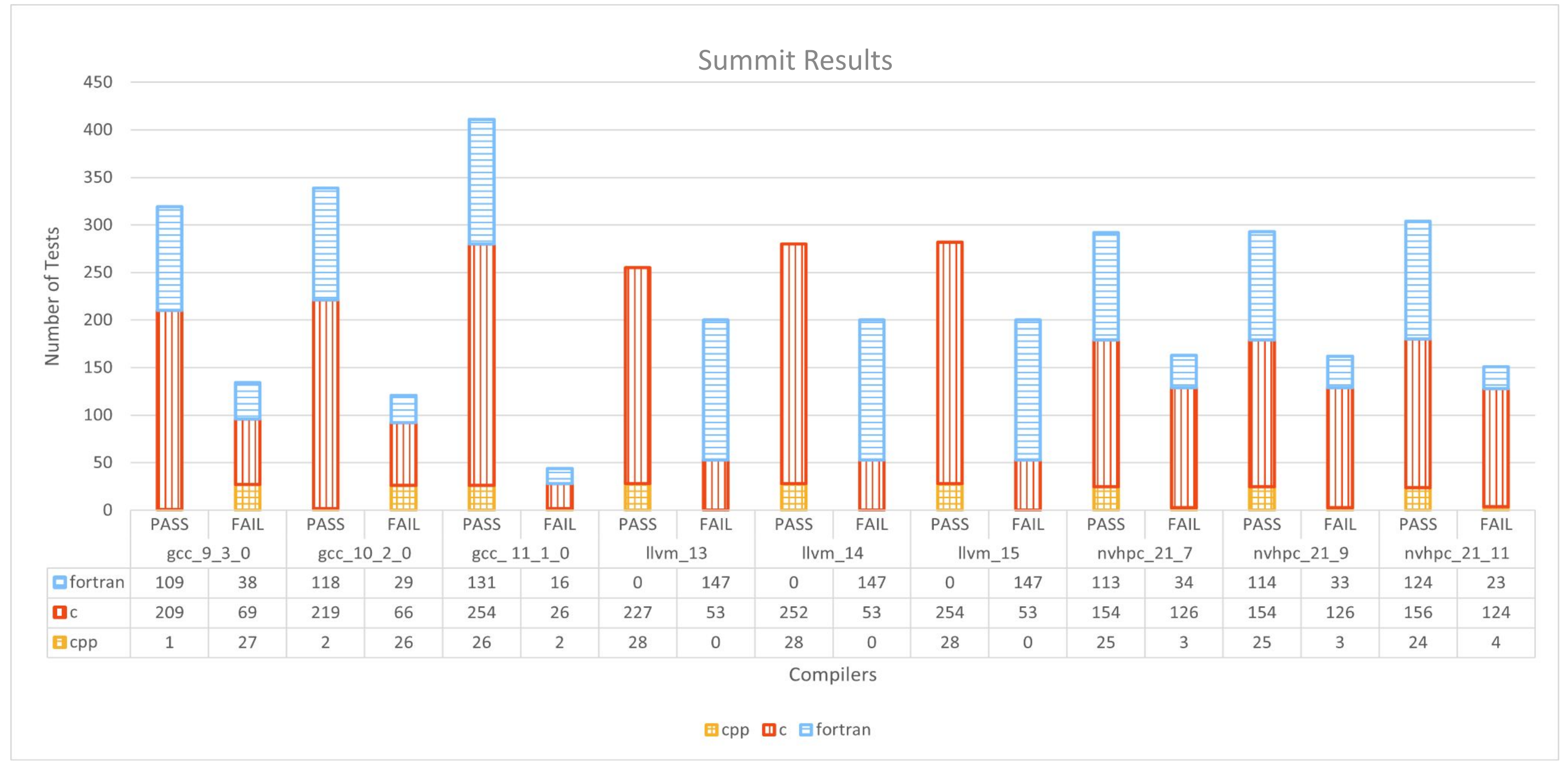
RESULTS ON PERLMUTTER, SUMMIT & CRUSHER



	GCC 11.2.0		CLANG 15.0.0		NVC 21.11		CCE 14.0.1	
	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL
fortran	89	83	0	0	98	74	119	53
c	177	83	173	87	92	168	139	121
cpp	24	4	26	2	21	7	0	28

Figure 2: Results for GCC, Clang, NVC & Cray on NERSC's Perlmutter system.

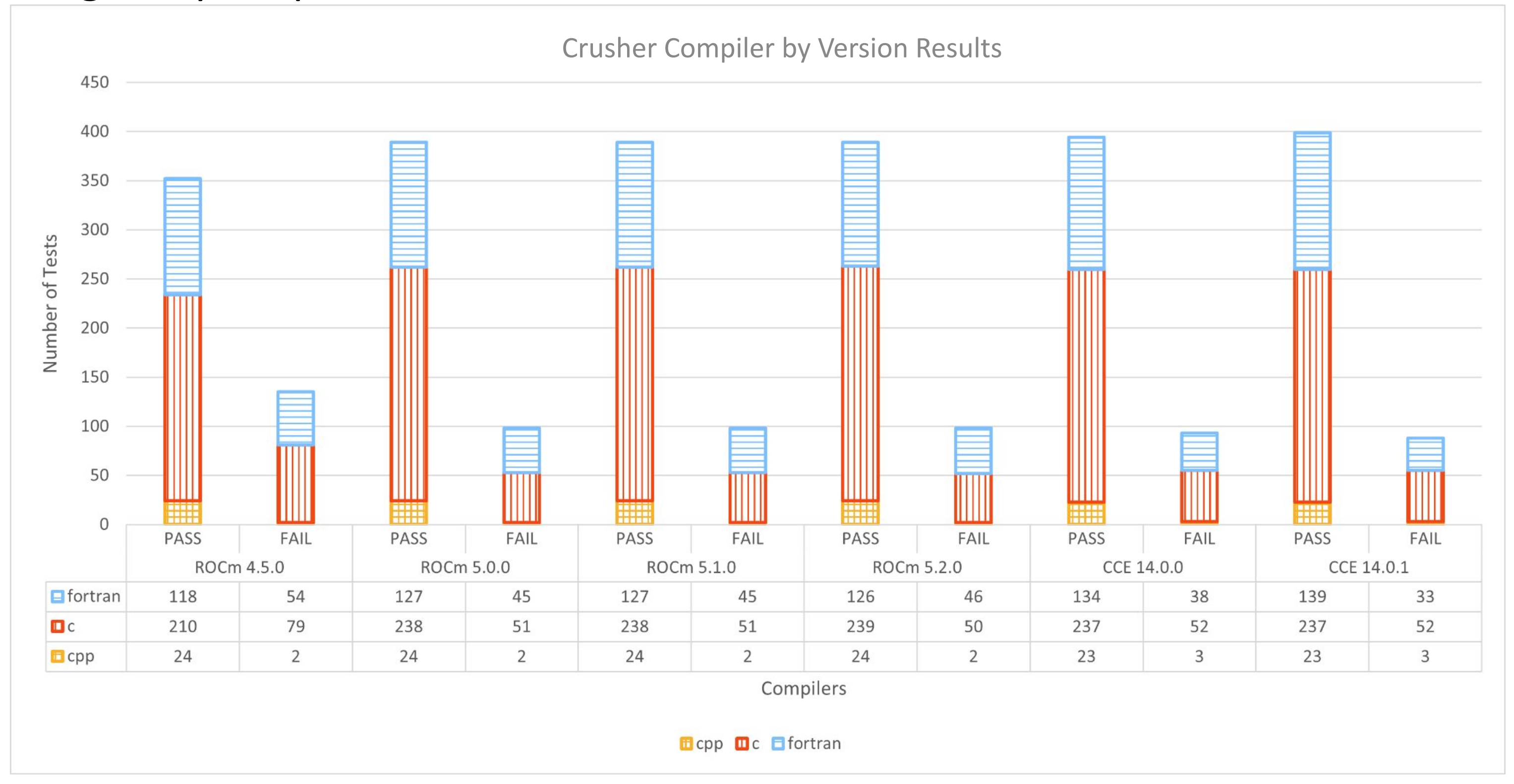
Figure 2 shows GCC & Clang perform best on Perlmutter, while NVC performs the worst, with more failing tests than passing. This is interesting, as the NVHPC compiler performs much better on other systems, such as Summit (Figure 3). It is important to note that LLVM's Clang does not include a fortran compiler.



	gcc_9_3_0		gcc_10_2_0		gcc_ 11_1_0		llvm_13		llvm_14		llvm_15		nvhpc_21_7		nvhpc_21_9		nvhpc_21_11	
	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL
fortran	109	38	118	29	131	16	0	147	0	147	0	147	113	34	114	33	124	23
c	209	69	219	66	254	26	227	53	252	53	254	53	154	126	154	126	156	124
cpp	1	27	2	26	26	2	28	0	28	0	28	0	25	3	25	3	24	4

Figure 3: Results for various Gnu's GCC, LLVM's Clang & Nvidia's NVHPC compiler versions on the ORNL Summit system.

Figure 3 shows GCC performing the best on the Summit system. Compared to Perlmutter, GCC has over 100 more passing tests on the system, despite it using a slightly downgraded version of the compiler. In contrast to perlmutter, LLVM's clang compiler performs the worst on Summit.



	ROCm 4.5.0		ROCm 5.0.0		ROCm 5.1.0		ROCm 5.2.0		CCE 14.0.0		CCE 14.0.1	
	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL
fortran	118	54	127	45	127	45	126	46	134	38	139	33
c	210	79	238	51	238	51	239	50	237	52	237	52
cpp	24	2	24	2	24	2	24	2	23	3	23	3

Figure 4: Results for ROCm compiler & Cray CCE compiler on ORNL's Crusher system.

Figure 4 shows results on Crusher, which is a pre-exascale system, performs quite well with both CCE & ROCm compiler. Compared to Perlmutter, the best-performing run on Crusher has around 50 more passing tests. It is interesting to note that previous versions of CCE, such as version 13.0.0, cannot run OpenMP code as it requires dependencies from both ROCm 5 & ROCm 4.

CONCLUSIONS

- In total, we have so far written 289 C, 26 C++ and, 172 Fortran tests
- The V&V suite targets up to OpenMP 5.2 specification
- The suite runs on pre-exascale systems (spock, crusher), Perlmutter and Argonne JLSE test beds

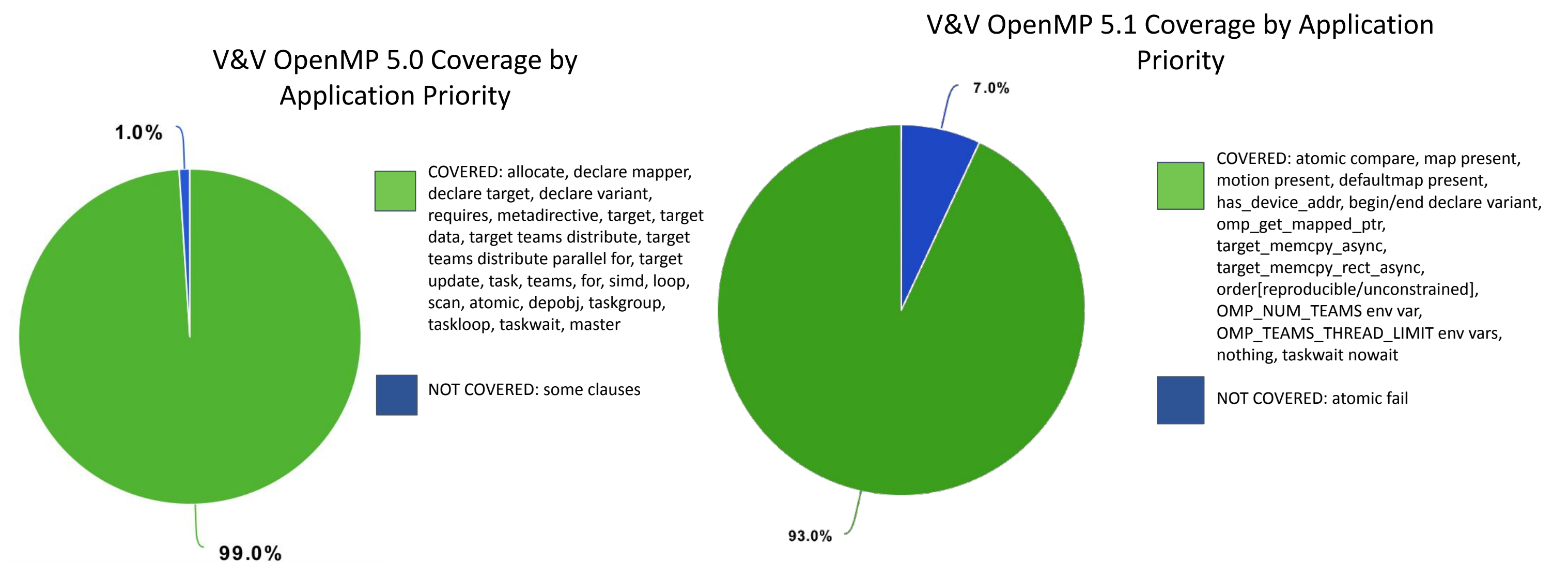
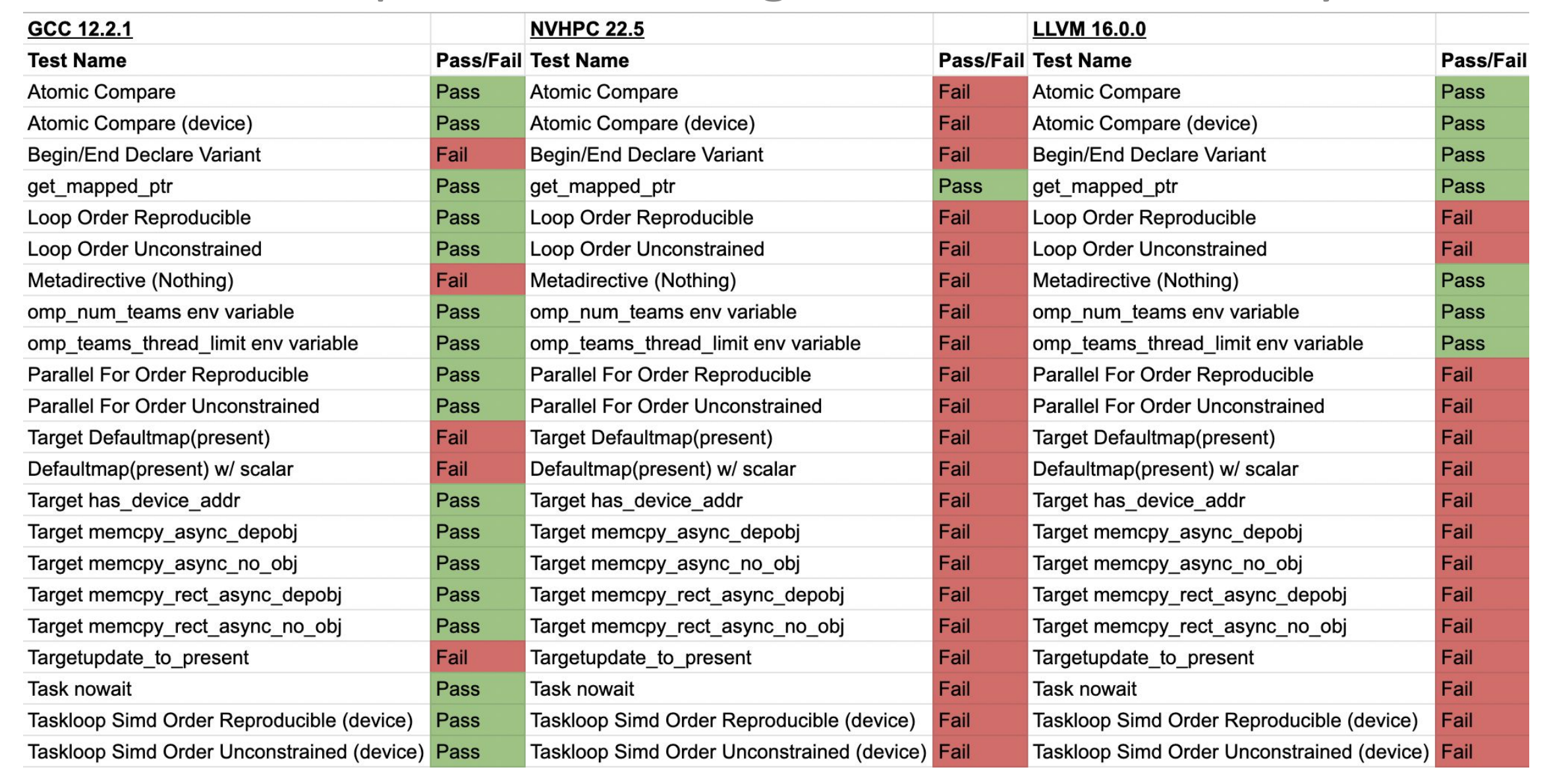


Figure 5: Pie charts showing percentage of testsuite's implementation of directives marked as "Medium" priority or higher by Application Developers at Oak Ridge National Laboratory.



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

Figure 6: Table displaying "medium" or higher priority tests implemented by GCC 12.1.1, NVHPC 22.5 & LLVM 16.

Figure 5 shows that our suite has implemented the majority of 5.0 & 5.1 features of high priority. Figure 6 illustrates that GCC has by far implemented more key features in OMP 5.1 compared to NVHPC & LLVM.

FUTURE WORK

- Working on new tests for the latest OpenMP specification (5.2)
- Expand our Fortran tests so that more C/C++ tests have Fortran counterparts
- Running/Testing the suite on more machines to further test the specification

CALLS TO ACTION

GitHub V&V repository (https://github.com/SOLLVE/sollve_vv) is open for download, logging issues, suggesting tests, etc.

Scan QR code to see results!

ACKNOWLEDGEMENTS

Thanks to OLCF: 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.

Thanks to ECP: This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Thanks to NERSC: This researched 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