

OpenACC Validation and Verification Testsuite

Sunita Chandrasekaran, UDEL, BNL (scandra@udel.edu)

Aaron Jarmusch and Christian Munley, UDEL
Computational Research and Programming Lab

Project Members:

UDEL: Aaron Liu, Will Gunter, Daniel Horta, Vaidhyanathan
Ravichandran

ORNL: Joel Denny

OpenACC Validation & Verification Testsuite

Goal

Create unit tests to
validate & verify
compilers'
implementation of
the OpenACC
specification



Revealing ambiguities in the OpenACC Specification



Determining missing implementation of a feature



Highlighting unmentioned restriction of a feature



Evaluating implementations for multiple target
platforms



Identifying and reporting compiler bugs

Jarmusch, A. M., Liu, A., Munley, C., Horta, D., Ravichandran, V., Denny, J., & Chandrasekaran, S. (2022). Analysis of Validating and Verifying OpenACC Compilers 3.0 and Above, 2022 Workshop on Accelerator Programming Using Directives (WACCPD), (pp. 1-10), IEEE.

Friedline, K. Chandrasekaran, S. Lopez, Graham M., Hernandez, O. OpenACC 2.5 Validation Testsuite targeting multiple architectures. (pp. 557-575), 2nd (P³MA) co-located with ISC, Germany, 2017. Springer International Publishing

Links to resources

- Open Source V&V Suite
 - <https://github.com/OpenACCUserGroup/OpenACCV-V>
- Website with results populated
 - <https://crpl.cis.udel.edu/oaccvv/>
- Example guide OpenACC
 - <https://github.com/OpenACC/openacc-examples>
- Practice codes
 - <https://github.com/Vaidh10/OpenACC-Practicecodes>

Status of OpenACC Testsuite Coverage

94 %

OVERALL coverage up to OpenACC
Version 3.3

84% - OpenACC Version 3.3
100% - OpenACC Version 3.2
93% - OpenACC Version 3.1
95% - OpenACC Version 3.0
100% - OpenACC Version 2.7

Results page:
<https://crpl.cis.udel.edu/oaccvv/results/>

439 C tests

452 C++ tests

440 Fortran
tests

Results

These results were last reviewed September 14, 2023

To get a quick overview of the results, please refer to the Summary tab. For in-depth analysis of runtime errors and other details, please check out the Results tab. The filter functionality only works on the Results tab.

Filter results	Search	Compiler results (CR) Choose from ▾	Runtime results (RR) Choose from ▾	Compiler Choose from ▾	Language Choose from ▾	System Choose from ▾
----------------	--------	---	--	----------------------------------	----------------------------------	--------------------------------

Summary Results

#	Test Name	System Name	NVC 23_1 CR	NVC 23_1 RR	GCC 12_2 CR	GCC 12_2 RR	Cray 15_0_0 CR	Cray 15_0_0 RR	Clacc #4879e9 CR	Clacc #4879e9 RR
1	acc_async_test.c	Perlmutter	Pass	Pass	-	-	-	-	-	-
2	acc_async_test.c	Crusher	-	-	Fail	Fail	-	-	-	-
3	acc_async_test.c	Summit	-	-	Fail	Fail	-	-	-	-

Example - gang dimensions

```

for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        arr1[i][j] = rand() / (real_t)(RAND_MAX / 10);
        arr2[i][j] = arr1[i][j] + 1;
    }
}
#pragma acc parallel num_gangs(n,n)
#pragma acc loop gang(dim:2)
for (int i = 0; i < n; i++)
{
    #pragma acc loop gang(dim:1)
    for (int j = 0; j < n; j++)
    {
        arr1[i][j] = arr1[i][j] + 1;
    }
}
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        if (fabs(arr1[i][j] - arr2[i][j]) > PRECISION) {
            err = 1;
        }
    }
}

```

775
776
777
778
779
780
781
782
783
784

- Allowed three dimensions of gang parallelism:
 - Defined multiple levels of *gang-redundant* and *gang-partitioned* execution modes. See Section 1.2
 - Allowed multiple values in the **num_gangs** clauses on the **parallel** construct. See Section 2.5.10.
 - Allowed a **dim** argument to the **gang** clause on the **loop** construct. See Section 2.9.2.
 - Allowed a **dim** argument to the **gang** clause on the **routine** directive. See Section 2.15.1.
 - Changed the launch event information to include all three gang dimension sizes. See Section 5.2.2.

Infrastructure Overview

Git Clone

<https://github.com/OpenACCUserGroup/OpenACCV-V.git>

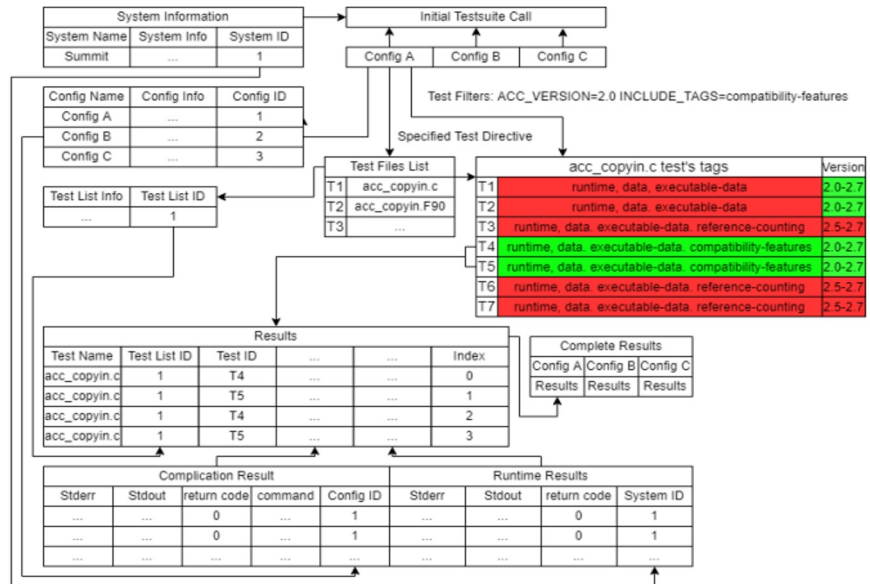
Edit config

Compiler, flags, output format, conditional compilation, etc

Run the python infrastructure

```
python3 infrastructure.py -c=<config_input_file> -o=<output_file>
```

Fig. 1: Overview of the infrastructure



Change the Config

OpenACC-V / init_config.txt

chrismun fixed config file type

5be87aa · 5 months ago History

Code Blame 181 lines (139 loc) · 5.61 KB

Raw Copy Download Edit History

```
1 !Welcome to the OpenACC Validation Suite Configuration File
2 #You can add comments either with the '!' or '#' symbols if you would like.
3 !To this end, I will give examples of how to use each of the settings that are configurable in the config file.
4 #If you would like one to be active, uncomment it and customize it :)
5
6 !The first settings are the compiler settings. Just set this to whatever you want to be invoked as the compiler
7 !If you don't have these set up in your path, you can also give full paths to be used
8
9 !CC:gcc
10 !CPP:g++
11 !FC:gfortran
12
13 !In addition to these, you will want to add some flags.
14 !Some of the features in the infrastructure use C Pre-Processor directives, so please make sure that is enabled.
15 !If enabling them is impossible, the infrastructure will be unable to detect which portions of the tests are causing
16 !compilation/runtime errors.
17
18 !CCFlags:-fopenacc -cpp -lm -foffload='-lm'
19 !CPPFlags:-fopenacc -cpp -lm -foffload='-lm'
20 !FCFlags:-fopenacc -cpp -lm -foffload='-lm'
21
22 !Below are a few more things that should probably be specified. The script will do its best, but sometimes it will
23 !TestDir:/home/<user>/OpenACC-V/Tests
24 !BuildDir:/home/<user>/OpenACC-V/Build
```

Required: Compilers,
compiler flags, output
format

Optional: runAllTests,
conditional compilation,
etc

OpenACC NVC and GCC V&V results

nvhpc 23.11 on Perlmutter (A100s)

1331 Total Tests

1095 Tests Pass

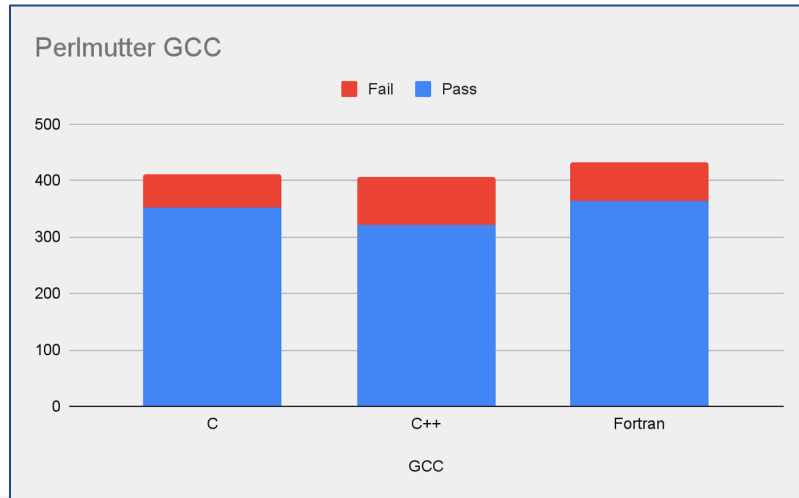
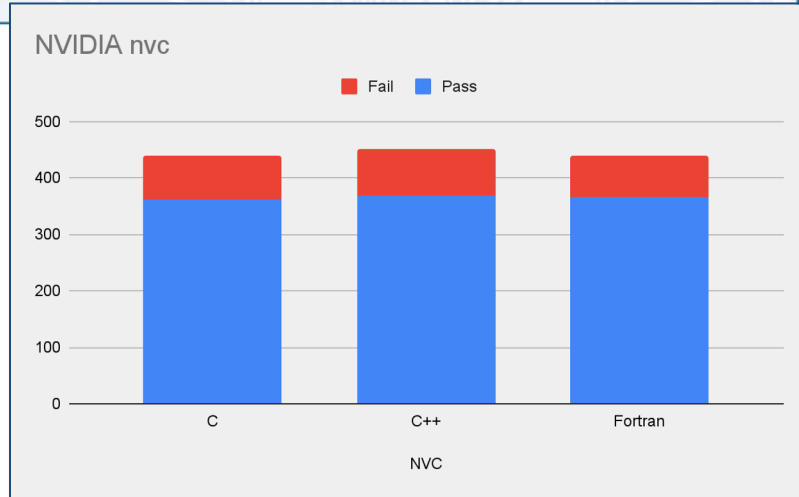
236 Tests Fail (Compiler/Runtime)

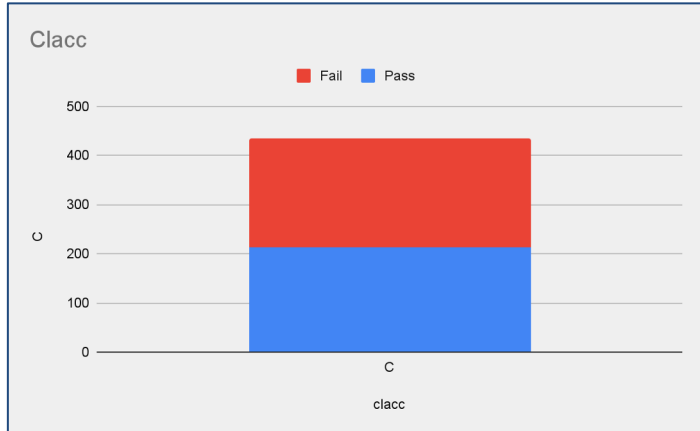
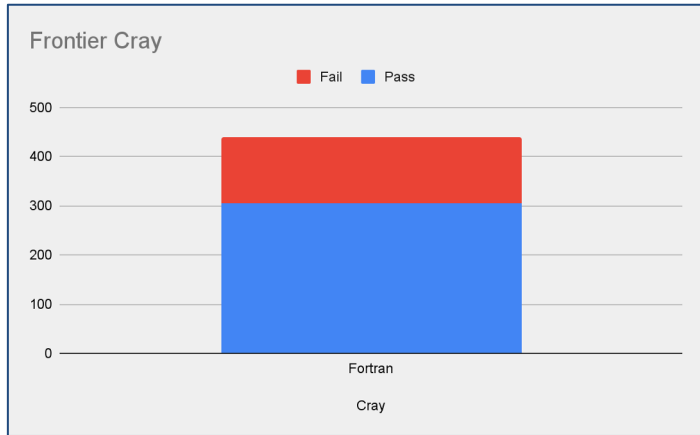
gcc 12.1.1 on Perlmutter (A100s)

1331 Total Tests

1051 Tests Pass

280 Tests Fails (Compiler/Runtime)





OpenACC V&V results

Cray Fortran 16.0.1 Frontier (AMD MI250x)

440 Total Tests

304 Tests Pass

136 Tests Fails (Compiler/Runtime)

Clacc Perlmutter (September) (NVIDIA A100s)

439 Total Tests

227 Tests Pass

212 Tests Fails (Compiler/Runtime)

Could we automate subsets of V&V tests?
How do we do so? Pros and Cons

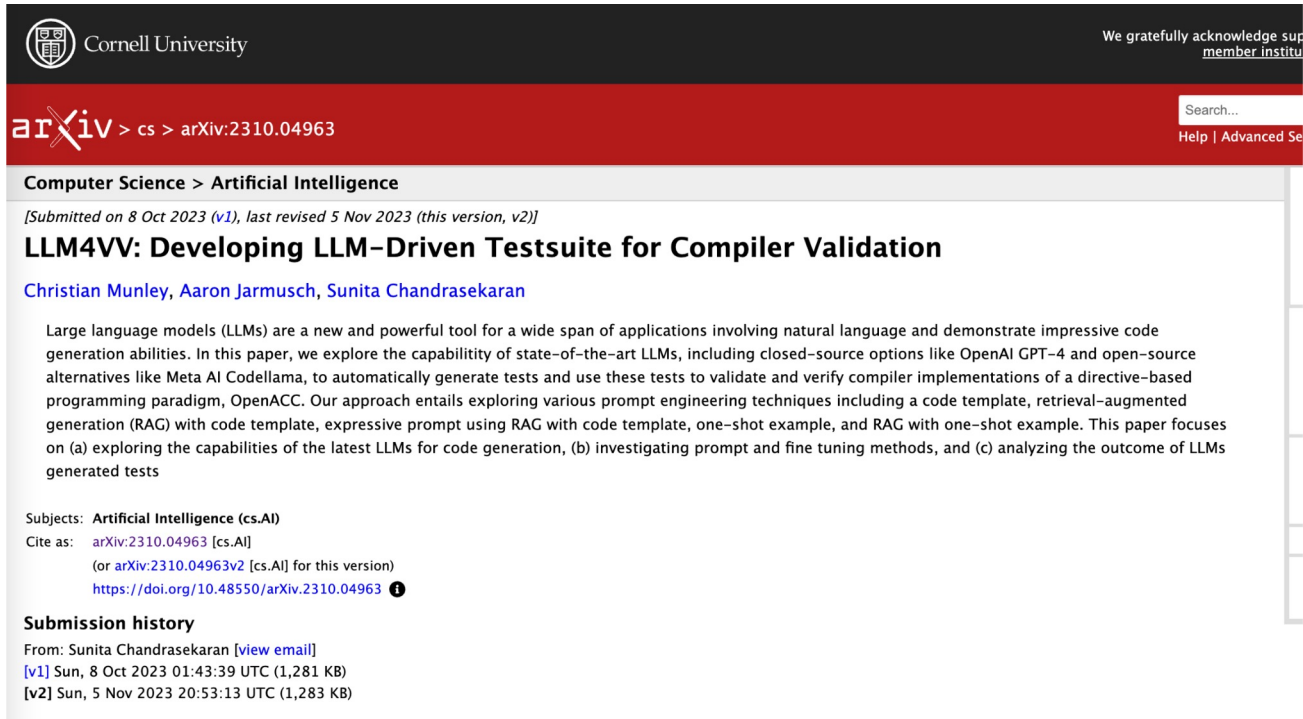
LLMs for compiler implementations' validation - Opportunities

- Standard specification evolves
- Programmers must learn and adapt: regular development
- Could we use LLMs to automate?
 - Programmers' time could be relieved from writing simple unit/functional tests and better spent writing corner/regression/unique test cases
 - Or could we use LLMs for these too? *We are NOT there yet!!*

LLMs for compiler implementations' validation - Challenges

- Prompts? Fine-tune? Train new LLMs?
- How to determine the quality of the LLM-generated tests?
- How do we tackle “hallucinations”?
- Watch out for carbon footprint when training LLMs
 - (GPT3 carbon emissions equivalent to driving 123 gasoline-powered cars for a year)

Preliminary Findings using LLMs



The screenshot shows the arXiv page for the paper "LLM4VV: Developing LLM-Driven Testsuite for Compiler Validation". The page header includes the Cornell University logo and a search bar. The paper title is prominently displayed, followed by the authors' names: Christian Munley, Aaron Jarmusch, and Sunita Chandrasekaran. The abstract discusses the use of Large Language Models (LLMs) for generating tests to validate compiler implementations. The page also includes subject tags, citation information, and a submission history section.

Cornell University

We gratefully acknowledge support from member institutions

arXiv > cs > arXiv:2310.04963

Search...
Help | Advanced Search

Computer Science > Artificial Intelligence

[Submitted on 8 Oct 2023 (v1), last revised 5 Nov 2023 (this version, v2)]

LLM4VV: Developing LLM-Driven Testsuite for Compiler Validation

Christian Munley, Aaron Jarmusch, Sunita Chandrasekaran

Large language models (LLMs) are a new and powerful tool for a wide span of applications involving natural language and demonstrate impressive code generation abilities. In this paper, we explore the capability of state-of-the-art LLMs, including closed-source options like OpenAI GPT-4 and open-source alternatives like Meta AI Codellama, to automatically generate tests and use these tests to validate and verify compiler implementations of a directive-based programming paradigm, OpenACC. Our approach entails exploring various prompt engineering techniques including a code template, retrieval-augmented generation (RAG) with code template, expressive prompt using RAG with code template, one-shot example, and RAG with one-shot example. This paper focuses on (a) exploring the capabilities of the latest LLMs for code generation, (b) investigating prompt and fine tuning methods, and (c) analyzing the outcome of LLMs generated tests

Subjects: **Artificial Intelligence (cs.AI)**

Cite as: [arXiv:2310.04963](https://arxiv.org/abs/2310.04963) [cs.AI]
(or [arXiv:2310.04963v2](https://arxiv.org/abs/2310.04963v2) [cs.AI] for this version)
<https://doi.org/10.48550/arXiv.2310.04963> ⓘ

Submission history

From: Sunita Chandrasekaran [[view email](#)]
[v1] Sun, 8 Oct 2023 01:43:39 UTC (1,281 KB)
[v2] Sun, 5 Nov 2023 20:53:13 UTC (1,283 KB)

Testsuite Generation for OpenACC w/ LLMs

- OpenAI GPT-3.5, GPT-4
- Meta's Codellama-34B-Instruct, Phind-Codellama-34B-v2
- Prompt engineering:
 - **Prompts** built from table of contents of the specification
 - **Retrieval Augmented Generation (RAG), one-shot prompt**
- Fine-tuning
- Stages of analysis

Stages

- **Stage 1:** 95 prompts
 - built from spec. table of content, only in C
 - 5 testsuites generated per LLM (prompt methods, fine-tuning)
 - Each ran, recording compile/runtime fail or pass.
 - Goal: compare methods
- **Stage 2:** 351 prompts
 - C/C++/Fortran, permutations of compute construct clause tests
 - 1 testsuite per LLM
 - Goal: compare LLMs
- **Stage 3** - analyze correctness

Some outcomes

- Meta's Codellama-34b-Instruct – produced 41 passing tests out of 335
- Phind-Codellama-34b-v2 - produced 95 passing tests
- **OpenAI's GPT 4 - produced 109 passing**
- OpenAI's GPT-3.5-ft - produced 43 passing

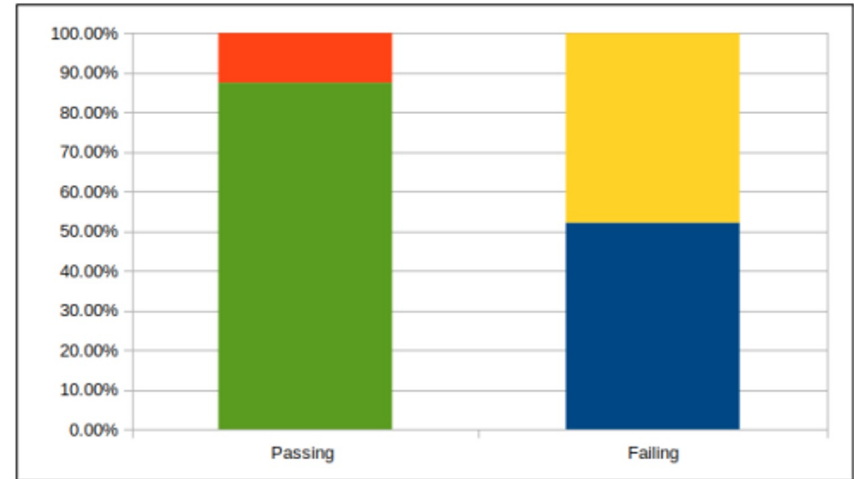
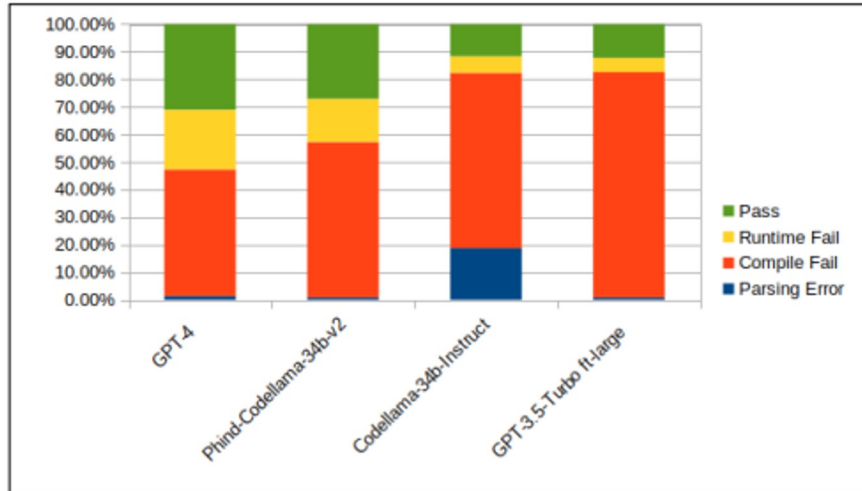


Figure 7: Stage 3 results displaying the analysis of a representative subset of generated tests by GPT-4. Green - True pass. Red - False Pass. Blue - Fails with issues in base language or compiler implementation. Yellow - Fails with incorrect OpenACC usage. The analysis shows that most passing tests are correct tests, whereas failing tests occur due to various reasons.

Findings

- Impressive performance on task, but room for improvement
- Benchmarks indicative of relative LLM performance
 - Task-specific benchmark will be useful
- Template > one-shot ...
- Performance is very sensitive to prompts
- Sometimes the test is right, sometimes it's not!
- Still need manual intervention

Improvements

- Domain specific fine-tuning
- Prompt
- Train HPC code model
- Task-specific benchmark

Summary

- Work in collaboration with OpenACC since 2017
- Compilers have evolved over a period of time
- Project is feedback-driven; input from vendors matters
- Time to think about which parts of the suite can be automated and how