COSC 6386
Program Analysis and Testing

Project Report

Open MPI Analysis

Group members

Kartik Kapoor (2092462)
Sushma Gangavarapu (2148457)
Bhuvana Chandra Adhugiri (2148341)

GitHub Repo link

https://github.com/kkapoor3/openmpi-analysis

Open MPI Introduction

Open MPI is a Message Passing Interface (MPI) library project combining technologies and resources from several other projects. The Open MPI Project is an open source MPI-2 implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers.

Building and testing OpenMPI

To build the open MPI, the following are the prerequisites before starting the open mpi build.

compilers - c compiler(Need a C compiler that supports C99)
GNU AutoTools - GNU m4, GNU Autoconf, GNU Automake, GNU Libtool

Steps to build and test OPENMPI

- **tar xf opcd openmpi-<version>**
    - Current version of open mpi is 4, where we have used the 4.1.3 version to build the project.
- **cd openmpi-<version>**
- **./configure --prefix=<path>**

- **./configure --prefix=<path>/gnu**
    - Using the above command .gcno and .gcda files are created which are used for code coverage.
- **make -j8 all**
- **make check**
    - The above command runs all the test cases and below are the some results of the test cases

```
============================================================================
Testsuite summary for Open MPI 4.1.3
============================================================================
# TOTAL: 8
# PASS:  8
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
```

```
make  check-TESTS
make[3]: Entering directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/datatype'
make[4]: Entering directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/datatype'
PASS: opal_datatype_test
PASS: unpack_hetero
PASS: checksum
PASS: position
PASS: position_noncontig
PASS: ddt_test
PASS: ddt_raw
PASS: ddt_raw2
PASS: unpack_ooo
PASS: ddt_pack
PASS: external32
PASS: large_data
PASS: partial
============================================================================
Testsuite summary for Open MPI 4.1.3
============================================================================
# TOTAL: 13
# PASS:  13
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[4]: Leaving directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/datatype'
make[3]: Leaving directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/datatype'
make[2]: Leaving directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/datatype'
Making check in util
make[2]: Entering directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/util'
make  opal_bit_ops opal_path_nfs bipartite_graph
make[3]: Entering directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/util'
make[3]: 'opal_bit_ops' is up to date.
make[3]: 'opal_path_nfs' is up to date.
make[3]: 'bipartite_graph' is up to date.
make[3]: Leaving directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/util'
make  check-TESTS
make[3]: Entering directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/util'
make[4]: Entering directory '/home/sushma/Desktop/new/openmpi-4.1.3/test/util'
PASS: opal_bit_ops
PASS: opal_path_nfs
PASS: bipartite_graph
============================================================================
```

Analysis Summary

Line Coverage = 5.7%
Function Coverage = 6.8%
Branch Coverage = 3.5%

Total Number of Test Files = 131
Total Number of Test Files with Assert Statements = 16
Percentage = 12%

Total Test Developers = 50

Total Number of Production Files = 2605
Total Number of Files with Assert Statements = 248
Total Number of Files with Debug Statements = 140
Percentage of Files with Assert Statements = 9%
Percentage of Files with Debug Statements = 5%

Analysis Report

After analyzing the Open MPI project, we can get some great insights into the state of testing in the project. From the analysis we did of the public GitHub repo of Open MPI, we found out that the **state of testing** in the project is not that great. The tests present in the project are not adequate at all. With the GCC code coverage report (Figure 1. GCC Code Coverage Report) we found out that the line coverage was only 5.7%, the branch coverage was only 3.5%, and function coverage was only 6.8%. The test coverage is very low and not at all adequate to find bugs and is not very helpful in debugging the project. This makes the developers' job very hard as they have to manually find bugs and spend more time developing their features and ensuring that they are bug-free before publishing. The lack of tests slows down the whole production process.

Upon investigation, we found out that the reason for the low line, branch and function coverage is simply the lack of tests written for the project. For perspective, there are 2605 production files in the project and only 131 test files. This translates to about one test file for every twenty production files. Out of these 131 test files, only 16 have ‘assert’ statements in them (Figure 2. Number of assert statements in Test Files), which is around 12% of the total test files. Out of these 16 files, there are only 5 files with more than 10 ‘assert’ statements and only 5 more with more than five ‘assert’ statements in them.

Due to the lack of test files, the developers had to write their own tests in their production files to test the build and the features. Out of 2605 production files, 248 have ‘assert’ statements for testing, which is about 9% of the total files, and 140 files have ‘debug’ statements, which is 5% of the total production files.

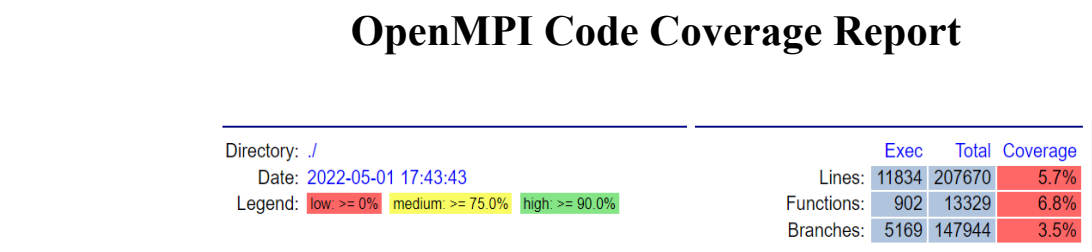

Figure 1. OpenMPI Code Coverage Report

We have deduced that there are five main disadvantages of developers writing their own tests in the production files. First, The developers will only write tests to test their own features and to find particular bugs. The developers might write tests only for unit testing and functional testing and not integration and system testing, which is not a good thing for a project. Second, if they write very specific tests, it will be difficult to find bugs and make the program bug free. Third, this will take up a lot of development time and the development process will slow down significantly. Fourth, since the tests are very specific and in the production files, they would not run during the testing phase of the project and some bugs might escape this stage and make their way into the working project. Fifth, any changes or additions to the tests will be difficult as they are written in their production files and not test files.

Test File Name	Number of Assert Statements
./ompi/test/asm/atomic_cmpset.c	69
./ompi/test/datatype/opal_datatype_test.c	34
./ompi/test/datatype/ddt_test.c	25
./ompi/test/datatype/ddt_raw.c	21
./ompi/test/class/opal_list.c	10
./ompi/test/datatype/ddt_raw2.c	8
./ompi/test/datatype/ddt_lib.c	7
./ompi/test/class/opal_pointer_array.c	7
./ompi/test/datatype/unpack_hetero.c	4
./ompi/test/datatype/opal_ddt_lib.c	4
./ompi/test/datatype/unpack_ooo.c	2
./ompi/test/support/support.c	2
./ompi/test/class/opal_value_array.c	1
./ompi/test/class/opal_cstring.c	1
./ompi/test/class/opal_lifo.c	1
./ompi/test/class/opal_fifo.c	1

Figure 2. Number of Assert statements in Test Files

One of the biggest reasons that we have identified for the lack of the number of test files is the lack of developers working on the test portion of the project. We have identified that there are only 50 developers who have contributed to the tests in the last 18 years and we do not think that it is a very good number. On average, there are less than three developers working on building the tests for the project each year. But of these 50 authors, there are only two who have more than 100 commits in the testing portion of the project, and only five others who have more than twenty commits (Figure 3. Top 15 Testing Contributors). So really there are less than 15 developers working on the testing portion of the project. And this is one of the biggest reasons

that the state of testing in the project is not that great. Within the last 18 years of development of the project there have been less than 15 people working on testing.

The developer support behind the testing files is not that great and that results in a poor testing state of the project. From analyzing the dates on which the testing files were added to the project (Figure 4. History of adding test files), we can see that during the initial development phase from 2004 to 2006, many test scripts were added. After that, over the years this practice slowed down quite a bit until 2021. During 2021-2022 we can see that more than 50 test files were added to the project, which is almost 40% of the total 131 test files. This made the testing state of the project slightly better, but not by much.

Tester Name	Number of Commits
Nathan Hjelm	129
Ralph Castain	115
Jeff Squyres	46
Brian Barrett	41
George Bosilca	36
Rainer Keller	22
David Daniel	20
Tim Woodall	18
Gilles Gouaillardet	13
luz.paz	10
Laura Casswell	10
Greg Koenig	9
bosilca	8
Shiqing Fan	6
Prabhanjan Kambadur	5

Figure 3. Top 15 Testing Contributors

By analyzing the commit history of the test files (Figure 5. Project Test Commit History) we can see that while the initial development of the project, there were a lot of commits to the project files between 2004 and 2006, but after that, the work on testing files pretty much stopped till 2015. Less than 5 new test files were added in 2015, and there were close to 50 commits. So, we can observe that some work was being done on those new files. In 2017 there were hardly any new files added, but more than 100 commits were made that year. We can safely assume that the developers were working on the existing tests. In 2021-2022 more than 50 new test files were added and the repo has more than 120 commits during that time. This tells us that the state of testing has become better in the last year or so, but it is still far from adequate.

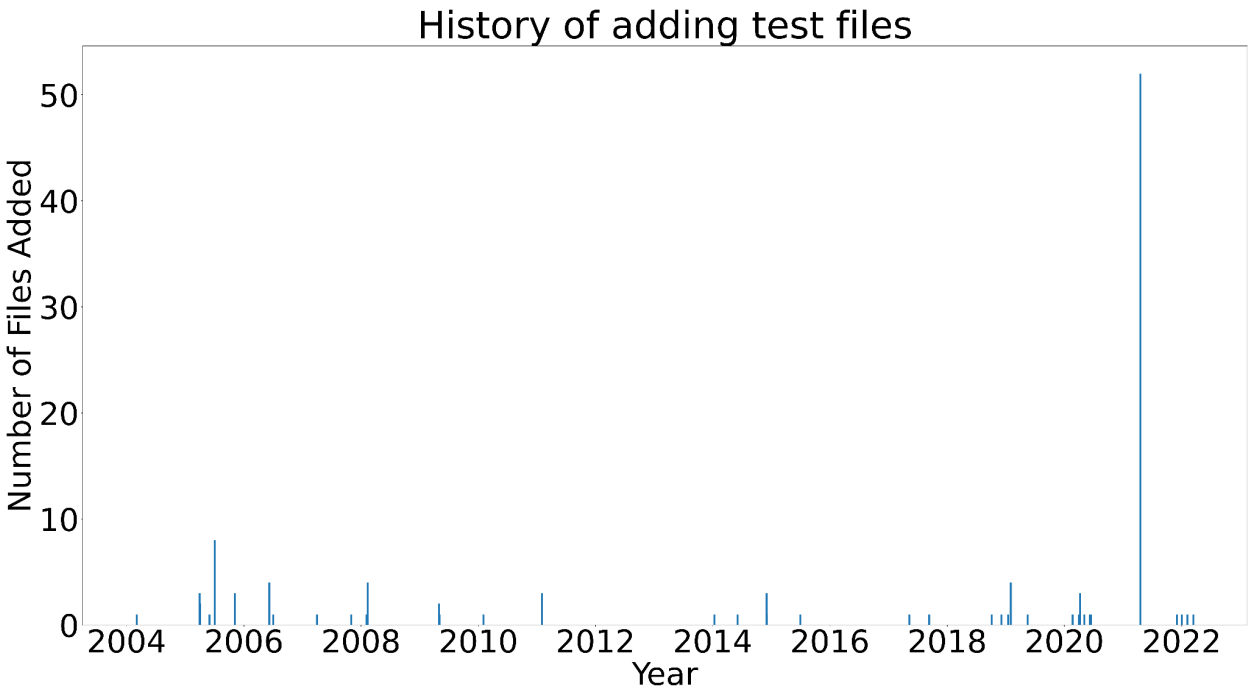

Figure 4. History of adding test files

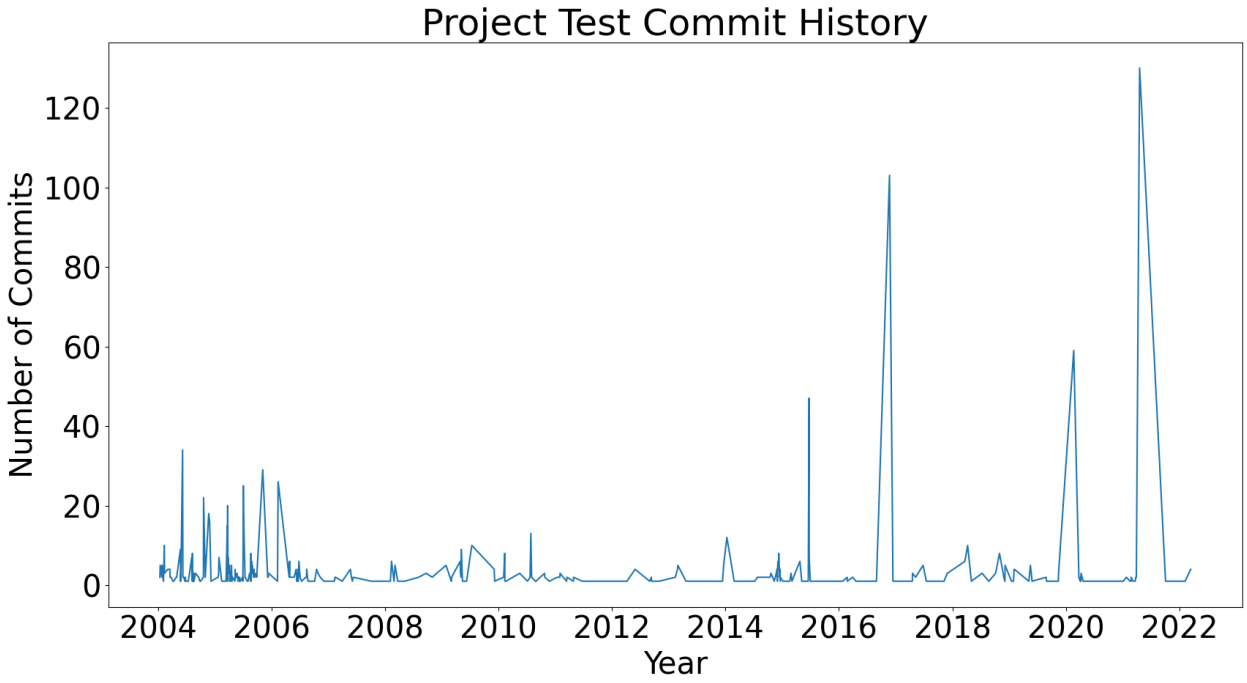

Figure 5. Project Test Commit History

The poor state of testing of the project affects the whole production cycle. The developers have to waste their time debugging their program, when they can utilize it to develop more features or optimize the old ones. The whole project takes around 40 minutes to build. Out of that 30 Minutes are used to build the features and another 10 minutes are used to run the tests. The tests take too much time to run even though the number of test files and the coverage is very low. The developers have to spend around 40 minutes each time they want to build the project and they have to wait for the test results after it is built. Given that the coverage of the test is very low this does not help the developers very much as it cannot provide a full picture of how their features are functioning and if they are truly bug free. So the developers have to waste much of their time in writing their own tests in addition to developing the features. But the developers of a feature are very limited in their scope of testing. They cannot properly implement the integration and system tests to determine if their feature integrates well within the project. This is a very big issue for the project and it might become a reason for failure of the project.

From the testers' perspective, developing new tests for the project will be challenging as there are very few existing tests and there are many production files and features for which they have to write new tests. For the whole project, the testers will have to write unit tests, functional tests, integration as well as system tests which will take a lot of time as there are so many production files. Also, they have to take into account the tests that are already written by the developers for their own features and ensure that those tests do not affect the new tests. Overall, improving the testing state of the project and increasing code coverage by writing new tests will be a very challenging task for the testers. In this case, the testers do not have to just write their own tests; they also have to take into account the existing developers' tests and probably get rid of those and start from scratch. This whole process slows down production. In these types of scenarios, test-driven development practice should be followed so that the test cases are written before the software is fully developed, and it is tested against the test cases repeatedly, making the software more robust.

Conclusion

After analyzing the whole OpenMPI project, we can safely say that the state of testing in the project is not very great. This is mainly due to the fact that there are only a few developers working on the testing part of the project. Due to this, the number of test files and the test coverage is very low. The lack of adequate testing slows down the project development cycle. According to our analysis, we find that the developers should follow the test-driven development practice and develop more test cases for the project, for it to be more stable.