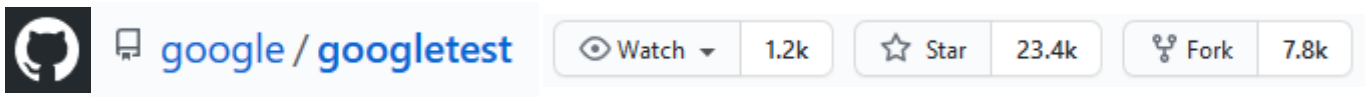
Testing HPC C++ software with GoogleTest:
adjusting the test framework for distributed-parallel tests using MPI

Melven Röhrig-Zöllner
Johannes Holke
German Aerospace Center (DLR), High-performance Computing

What is Googletest?



- Unit testing library for C++ codes (https://github.com/google/googletest)

- Offers an easy to use framework to test your code (and more)

- Features include
  - Value-parameterized tests
  - Type-parameterized tests
  - User-defined assertions
  - Many more

What is Googletest? Examples

Check if our library correctly computes the norm of the vector (0 0 0)

main.cpp

```cpp
#include "gtest/gtest.h"

int main(int argc, char** argv)
{
    ::testing::InitGoogleTest(&argc, argv);

    return RUN_ALL_TESTS();
}
```

test_vec.cpp

New test case

Check if norm(0) == 0

```cpp
TEST (t8_gtest_vec, normzero) {
  const t8_test_vec zero = {0, 0, 0};

  EXPECT_EQ (t8_vec_norm (zero), 0);
}
```

What is Googletest? Examples

Test passed

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from t8_gtest_vec
[ RUN      ] t8_gtest_vec.normzero
[       OK ] t8_gtest_vec.normzero (0 ms)
[----------] 1 test from t8_gtest_vec (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
```

Test failed

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from t8_gtest_vec
[ RUN      ] t8_gtest_vec.normzero
../../source/t8code/test/t8_gtest_vec.cxx:38: Failure
Expected equality of these values:
  t8_vec_norm (zero)
    Which is: 1
  0
[  FAILED  ] t8_gtest_vec.normzero (0 ms)
[----------] 1 test from t8_gtest_vec (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] t8_gtest_vec.normzero

 1 FAILED TEST
```

What is GoogleTest? Output of our phist test suite

```
[ RUN      ] DSymmMatTestWithUnalignedViews_163_30.read_matrices
[       OK ] DSymmMatTestWithUnalignedViews_163_30.read_matrices (0 ms)
[ RUN      ] DSymmMatTestWithUnalignedViews_163_30.A1_probe_symmetry
[       OK ] DSymmMatTestWithUnalignedViews_163_30.A1_probe_symmetry (1 ms)
[----------] 2 tests from DSymmMatTestWithUnalignedViews_163_30 (1 ms total)

[----------] 2 tests from ZSymmMatTestWithUnalignedViews_163_30
To avoid possible inconsistent ordering between matrices, we force sigma=1
 right now, see issue #225.
[ RUN      ] ZSymmMatTestWithUnalignedViews_163_30.read_matrices
[       OK ] ZSymmMatTestWithUnalignedViews_163_30.read_matrices (0 ms)
[ RUN      ] ZSymmMatTestWithUnalignedViews_163_30.A1_probe_symmetry
[GHOST] PE0 INFO at tsmttsm_plain_kernel() <tsmttsm_var2_plain__var_var.cpp:53>: In UNALIGN
[GHOST] PE1 INFO at tsmttsm_plain_kernel() <tsmttsm_var2_plain__var_var.cpp:53>: In UNALIGN
[GHOST] PE2 INFO at tsmttsm_plain_kernel() <tsmttsm_var2_plain__var_var.cpp:53>: In UNALIGN
[       OK ] ZSymmMatTestWithUnalignedViews_163_30.A1_probe_symmetry (1 ms)
[----------] 2 tests from ZSymmMatTestWithUnalignedViews_163_30 (1 ms total)

[----------] Global test environment tear-down
[==========] 14642 tests from 1096 test cases ran. (4888 ms total)
[  PASSED  ] 14642 tests.

  YOU HAVE 3562 DISABLED TESTS
```

What is Googletest? Macro Assertions

EXPECT_* - Reports failure, test continues

Use whenever possible

ASSERT_* - Reports failure, test aborts

Use when test would not be able to continue after failure (i.e. open file, initialize)

EXPECT/ASSERT_*	
EQ (A,B)	A == B
NEAR (A, B, eps)	|A-B| < eps
NE (A,B)	A != B
LT (A,B)	A < B
LE, GT, GE	<=, >, >=
TRUE (A)	A == true
FALSE (A)	A == false

What is MPI

- „Message Passing Interface“ (https://www.mpi-forum.org/)
- API for distributed memory parallelism
- Manages interaction of parallel processes and exchange of „messages“
- Backbone of many HPC libraries
- Multiple different implementations (OpenMPI, mpich, mpi4py, Intel MPI, etc.)

MPI example: hello world

```cpp
int main (int argc, char* argv[])
{
    int mpiret;
    int mpirank;

    mpiret = MPI_Init (&argc, &argv);
    CHECK_MPI (mpiret);

    mpiret = MPI_Comm_rank (MPI_COMM_WORLD, &mpirank);
    CHECK_MPI (mpiret);

    std::cout << "Hello from " << mpirank << std::endl;

    MPI_Finalize ();

    return 0;
}
```

Each proces has unique „rank“ 0, 1, 2, … NP-1

```
>  mpirun -np 4 ./mpi_helloword
Hello from 3
Hello from 2
Hello from 0
Hello from 1
```


MPI example: broadcast

```cpp
int somevalue = 0;
if (mpirank == 0) {
    somevalue = 42;
}

std::cout << mpirank << " has value: " << somevalue << std::endl;

MPI_Bcast (&somevalue, 1, MPI_INT, 0, MPI_COMM_WORLD);

std::cout << mpirank << " has value: " << somevalue << std::endl;
```

Rank 0

sends 1 integer at position „somevalue“

to all processes

```
> mpirun -np 4 ./mpi_bcast
1 has value: 0
2 has value: 0
3 has value: 0
0 has value: 42
0 has value: 42
1 has value: 42
2 has value: 42
3 has value: 42
```

MPI outline

- Point-to-point communication
- Reduction operations
- Gather and Scatter type operations
- etc.

MPI calls can be
- Collective – All processes must call the function.
- Blocking – The program can only continue when the MPI operation is finished.

⇨ Possibility of Deadlocks

```
if (mpirank == 0) {
    MPI_Bcast (&somevalue, 1, MPI_INT, 0, MPI_COMM_WORLD);
}
```

→ Process 0 will wait forever on the other processes!

We want to test our MPI parallelized libraries using GoogleTest!

Problem 1 – Communication after Assert

```
TEST (MPI_Test, BlockingCommunicationAfterAssert) {
    int mpirank;
    int mpiret;
    mpiret = MPI_Comm_rank (MPI_COMM_WORLD, &mpirank);
    ASSERT_EQ (mpiret, MPI_SUCCESS);

    ASSERT_EQ (0, mpirank);

    std::cout << mpirank << ": starting MPI communication\n";
    doMPICommunication ();
    std::cout << mpirank << ": finished MPI communication\n";
}
```

- Compute the MPI rank and check if no error occured.
- This will only pass on process 0. All other processes will abort.
- Trigger blocking MPI communication

Process 1

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from MPI_Test
[ RUN      ] MPI_Test.BlockingCommunicationAfterAssert
MPI_test.cpp:39: Failure
Expected equality of these values:
  0
  mpirank
    Which is: 1
[  FAILED  ] MPI_Test.BlockingCommunicationAfterAssert (0 ms)
[----------] 1 test from MPI_Test (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] MPI_Test.BlockingCommunicationAfterAssert

 1 FAILED TEST
```

Process 0

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from MPI_Test
[ RUN      ] MPI_Test.BlockingCommunicationAfterAssert
0: starting MPI communication
```

Solution to problem 1 – synchronized assertions

- Introduce new macros `ASSERT_*_MPI`
- synchronize assertion result using MPI communication
- If one process fails the assertion, all will fail the assertion

```
TEST (MPI_Test, BlockingCommunicationAfterAssert) {
    int mpirank;
    int mpiret;
    mpiret = MPI_Comm_rank (MPI_COMM_WORLD, &mpirank);
    ASSERT_EQ_MPI (mpiret, MPI_SUCCESS);

    ASSERT_EQ_MPI (0, mpirank);

    std::cout << mpirank << ": starting MPI communication\n";
    doMPICommunication ();
    std::cout << mpirank << ": finished MPI communication\n";
}
```

Process 0

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from MPI_Test
[ RUN      ] MPI_Test.BlockingCommunicationAfterAssert
MPI_test.cpp:44: Failure

[  FAILED  ] MPI_Test.BlockingCommunicationAfterAssert (0 ms)
[----------] 1 test from MPI_Test (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] MPI_Test.BlockingCommunicationAfterAssert

 1 FAILED TEST
```

Solution to problem 1 – implementation details

- GTest uses an “AssertionResult” class:
  - New variable “global”
  - If global → result is synchronized
- Basic idea:
  - “AssertionResult” needs boolean arithmetic (e.g. negate results, check for failure, …)

→ tri-state logic instead of boolean:

A	not A	Test result (A)
all true	all false	success
all false	all true	failure
mixed	mixed	failure

```
  template <typename T>
  explicit AssertionResult(
      const T& success, bool global = false,
      typename std::enable_if<
          !std::is_convertible<T, AssertionResult>::value>::type*
      /*enabler*/
      = nullptr)
      : success_(success), globalResultsDiffer_(false) {
#if GTEST_HAS_MPI
        if( global )
          globalResultsDiffer_ = !boolIdenticalOnMPIprocs(success_);
#endif
      }

#if defined(_MSC_VER) && _MSC_VER < 1910
  GTEST_DISABLE_MSC_WARNINGS_POP_()
#endif

  // Assignment operator.
  AssertionResult& operator=(AssertionResult other) {
    swap(other);
    return *this;
  }

  // Returns true if and only if the assertion succeeded.
  operator bool() const { return success_ && !globalResultsDiffer_; }
```

Problem 2 – Processes do different things

```
TEST (MPI_Test, IndependentProcesses) {
    int mpirank, mpiret;
    mpiret = MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    ASSERT_EQ_MPI(mpiret, MPI_SUCCESS);

    if( mpirank == 1 ) {
        ASSERT_EQ(42, 0);
    }
}
```

→ Only happens on process 1 and not on process 0

Process 0

```
> mpirun -np 2 ./test1
```

...

```
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from MPI_Test
[ RUN      ] MPI_Test.IndependentProcesses
[       OK ] MPI_Test.IndependentProcesses (13 ms)
[----------] 1 test from MPI_Test (13 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (13 ms total)
[  PASSED  ] 1 test.
```

→ **Test seems to succeed!**
(“false success”)

Solution to problem 2 – synchronize final test result

- Synchronize test result using MPI communication
- If one process fails the test, all will fail
- Implementation similar as for problem 1…

Process 0

```
> mpirun -np 2 ./test1
...
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from MPI_Test
[ RUN      ] MPI_Test.IndependentProcesses
[  FAILED  ] MPI_Test.IndependentProcesses (1 ms)
[----------] 1 test from MPI_Test (2 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (2 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] MPI_Test.IndependentProcesses

 1 FAILED TEST
```

Problem 3 – Output

- Usually, the output of all processes is mixed up:

```
> mpirun -np 2 ./test1
Running main() from gtest_main.cc
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from MPI_Test
[ RUN      ] MPI_Test.IndependentProcesses
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from MPI_Test
[ RUN      ] MPI_Test.IndependentProcesses
/home/melven/googletest_examples/test1.cpp:11: Failure
      Expected: 42
To be equal to: 0
[  FAILED  ] MPI_Test.IndependentProcesses (0 ms)
[  FAILED  ] MPI_Test.IndependentProcesses (1 ms)
[ RUN      ] MPI_Test.simple_fail
/home/melven/googletest_examples/test1.cpp:16: Failure
      Expected: 2
To be equal to: 1
[ RUN      ] MPI_Test.simple_fail
/home/melven/googletest_examples/test1.cpp:16: Failure
      Expected: 2
To be equal to: 1
```

⋮

⋮

```
[  FAILED  ] MPI_Test.simple_fail (0 ms)
[  FAILED  ] MPI_Test.simple_fail (1 ms)
[----------] 2 tests from MPI_Test (2 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (2 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 2 tests, listed below:
[  FAILED  ] MPI_Test.IndependentProcesses
[  FAILED  ] MPI_Test.simple_fail

 2 FAILED TESTS
[----------] 2 tests from MPI_Test (1 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (1 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 2 tests, listed below:
[  FAILED  ] MPI_Test.IndependentProcesses
[  FAILED  ] MPI_Test.simple_fail

 2 FAILED TESTS
```

Solution to problem 3 – only output from process 0

- Per default, only print output on process 0
- Same applies to the XML output

→ Can be overwritten by the user…

```
if( rank == 0 ) {
  // Configures listeners for default output. Initializing it here
  // allows us to check if this is the MPI root process to prevent
  // duplicate output.
  listeners()->SetDefaultResultPrinter(new PrettyUnitTestResultPrinter);

  // Configures listeners for XML output. This makes it possible for users
  // to shut down the default XML output before invoking RUN_ALL_TESTS.
  ConfigureXmlOutput();
} else {
  listeners()->SetDefaultResultPrinter(NULL);
}
```

Processes 0 and 1

```
> mpirun -np 2 ./test1
Running main() from gtest_main.cc
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from MPI_Test
[ RUN      ] MPI_Test.IndependentProcesses
[  FAILED  ] MPI_Test.IndependentProcesses (2 ms)
[ RUN      ] MPI_Test.simple_fail
/home/melven/googletest_examples/test1.cpp:17: Failure
      Expected: 2
To be equal to: 1
[  FAILED  ] MPI_Test.simple_fail (0 ms)
[----------] 2 tests from MPI_Test (2 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (2 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 2 tests, listed below:
[  FAILED  ] MPI_Test.IndependentProcesses
[  FAILED  ] MPI_Test.simple_fail

 2 FAILED TESTS
```

Further changes in the background

- Check / call MPI initialization in (`MPI_Init` / `MPI_Finalize`)

- Use dedicated MPI communicator
  → unit test communication does not interfere with user communication

- Perform file-IO on the master only (for now) + communicate result:
  - processes can run on different nodes
  - processes on the same node “see” the filesystem differently
    (proc. 0 creates folder, not immediately visible on proc. 1 – OS may only ensures this for one process!)

Not a solution

- There are multiple repos / stackoverflow-answers / etc. that only solve “problem 3” (mixed-up output)

- **This is not sufficient!**
  - Problem 1 → dead-locks
  - Problem 2 → false success

(we have seen this in bigger software projects in use for years!)

→ You cannot test parallel software without thinking about the constraints / implications of the parallelization!

Difficulties & constraints of parallel testing with MPI

- MPI built for high-performance…
- No detection of communication errors (dead-locks, mismatching messages / etc.)
  → need to run unit tests with a tool, e.g. must: https://itc.rwth-aachen.de/must/
- No death tests
- User must decide where to use local vs. global assertions!
  (`ASSERT_EQ` vs. `ASSERT_EQ_MPI`)

Conclusion

- Please use and cite our MPI GoogleTest version: https://github.com/DLR-SC/googletest_mpi

- It's been in use in our institute since 2014 in several software projects:
  - Sparse linear algebra (phist, spliss)
  - Helicopter simulation (VAST)
  - CFD mesh management (FSDM, t8code)
  - Multi-linear/tensor algebra (pitts)

- What could be done to improve googletest_mpi?
  - Show detailed error message from first failing process
  - Show ranks of failing processes / how many ranks failed
  - Add "`_MPI`" variant for all `EXPECT_` / `ASSERT_` statements