Automatic detection of MPI assertions

Tim Jammer, Christian Iwainsky, Christian Bischof
Scientific Computing & HKHLR

tim.jammer@tu-darmstadt.de

https://github.com/tudasc/mach

C3PO'20: Compiler-assisted Correctness Checking and Performance Optimization for HPC

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- The 2019 draft specification of MPI [Mes19] defines new communicator info hints:
  - mpi_assert_allow_overtaking
  - mpi_assert_exact_length
  - mpi_assert_no_any_tag
  - mpi_assert_no_any_source
- If these assertions are given a more optimized implementation man be used

Scientific Computing Hessisches Kompetenzzentrum für Hochleistungsrechnen

Motivation

- The 2019 draft specification of MPI [Mes19] defines new communicator info hints:
  - mpi_assert_allow_overtaking
  - mpi_assert_exact_length
  - mpi_assert_no_any_tag
  - mpi_assert_no_any_source
- If these assertions are given a more optimized implementation man be used

- We propose to automatically detect if these assertions hold using a Clang/LLVM compiler pass

$\Rightarrow$ This saves the developers effort to manually check if these assertions hold

Outline

allow_overtaking assertion

exact_length assertion

no_any_tag and no_any_source assertion

Evaluation

Summary

Outline

allow_overtaking assertion

```
if (rank == 0){
 int buffer;
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
     MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
 // do something with message 1
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
     MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer = 1;
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
     MPI_COMM_WOLRD);
 // prepare message 2
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
     MPI_COMM_WOLRD);
}
```

Listing 1: Standard-conform MPI code snippet

- The messages must be received in the same order as they where sent

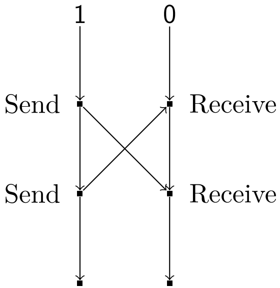


Figure: Messages overtaking each other

allow_overtaking **assertion**

```
if (rank == 0){
 int buffer;
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
 // do something with message 1
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer = 1;
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD);
 // prepare message 2
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD);
}
```

Listing 1: Standard-conform MPI code snippet

- The messages must be received in the same order as they where sent

+ Ensuring the order of messages is one of the most costly phases of message matching [SDGB18]
+ If message overtaking is allowed, this phase may be entirely skipped

allow_overtaking **assertion analysis**

```
if (rank == 0){
 int buffer;
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
     MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
 // do something with message 1
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
     MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer = 1;
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
     MPI_COMM_WOLRD);
 // prepare message 2
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
     MPI_COMM_WOLRD);
}
```

Listing 2: Standard-conform MPI code snippet

⇒ We check if there is any pair of sending operations, where the messages might overtake each other

- A pair of sending operations is considered conflict-free if at least one of these conditions hold:
  1. they use a different communicator
  2. they use a different message tag
  3. they are sent to different target ranks
  4. there is no codepath between them
  5. they are separated by a synchronization of the processes

allow_overtaking **assertion analysis**


```
#define TAG 42
int buffer;
MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
    MPI_COMM_WOLRD);
// prepare message 2
MPI_Send(&buffer, 1, MPI_INT, 0, TAG + 1,
    MPI_COMM_WOLRD);
// prepare message 3
MPI_Send(&buffer, 1, MPI_INT, 1, TAG,
    MPI_COMM_WOLRD);
}
```

Listing 3: Non-conflicting sending operations

1. **they use a different communicator**
2. **they use a different message tag**
3. **they are sent to different target ranks**
4. there is no codepath between them
5. they are separated by a synchronization of the processes

- Often it is possible to statically prove a difference in tag, target or communicator
- We use LLVM's ScalarEvolution analysis for this purpose

allow_overtaking assertion analysis

```
int buffer;
if (rank % 2 == 0)
{
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD)
 MPI_Recv(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IGNORE)
}else{
 int to_send = buffer;
 MPI_Recv(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IGNORE)
 MPI_Send(&to_send, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD)
}
```

Listing 4: Only one sending operation will be executed

1. they use a different communicator
2. they use a different message tag
3. they are sent to different target ranks
4. **there is no codepath between them**
5. they are separated by a synchronization of the processes

- If there does not exist a path in the control flow graph between the two operations, they are conflict free
- Note that an operation can conflict with itself e.g. in a loop

allow_overtaking **assertion analysis**

```
if (rank == 0){
 int buffer = 1;
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
 MPI_Barrier(MPI_COMM_WORLD);
 MPI_Recv(&buffer, 1, MPI_INT, 1, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer;
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD);
 MPI_Barrier(MPI_COMM_WORLD);
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD);
}
```

Listing 5: Communication is split by synchronization

1. they use a different communicator
2. they use a different message tag
3. they are sent to different target ranks
4. there is no codepath between them
5. **they are separated by a synchronization of the processes**

- If there is a barrier or an allreduce on all possible codepaths between them:
- We can assume that the target process has completed the matching receive
  - Otherwise there might be a deadlock
- For this analysis: consider non-blocking operations at corresponding wait

allow_overtaking **assertion analysis**

Recap

- Check all pairs of sending operations if
  1. they use a different communicator
  2. they use a different message tag
  3. they are sent to different target ranks
  4. there is no codepath between them
  5. they are separated by a synchronization of the processes

$\Rightarrow$ If for all pairs at least one condition hold one can specify the assertion

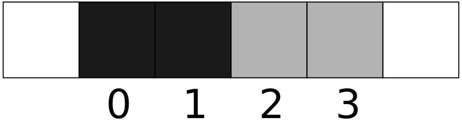
Outline

exact_length **assertion**

```
if (rank == 0){
 int buffer[4];
 MPI_Recv(&buffer, 4, MPI_INT, 1, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer[2];
 MPI_Send(&buffer, 2, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD);
}
```

Listing 6: Standard conform MPI code snipped

Figure: Illustration of the receive buffer



- Receiving a message into a shorter buffer is not allowed
- Supplying MPI_Recv with a shorter buffer than indicated by type × count is not allowed
- \+ MPI forbids the modification of buffer[2-3] in this example
- \+ The possibility to overwrite these excess buffer locations will allow for some optimizations

exact_length **assertion**

```
if (rank == 0){
 int buffer[4];
 MPI_Recv(&buffer, 4, MPI_INT, 1, TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer[2];
 MPI_Send(&buffer, 2, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD);
}
```

Listing 6: Standard conform MPI code snipped

- Receiving a message into a shorter buffer is not allowed
- Supplying MPI_Recv with a shorter buffer than indicated by type $\times$ count is not allowed

+ MPI forbids the modification of buffer[2-3] in this example
+ The possibility to overwrite these excess buffer locations will allow for some optimizations

- Implementation as a proof of concept that static analysis can check for this assertion
- Grouping all Send/Recv operations by message tag
  - $\Rightarrow$ If all operations within each group use the same message size, the assertion may be specified

Outline

no_any_tag / no_any_source **assertion**

```
if (rank == 0){
 int buffer;
 MPI_Recv(&buffer, 1, MPI_INT,
      MPI_ANY_SOURCE, MPI_ANY_TAG,
      MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer;
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
      MPI_COMM_WOLRD);
}
```

Listing 7: Standard conform MPI code snipped

- MPI_ANY_TAG and MPI_ANY_SOURCE s to implement non deterministic communication schemes

+ Not using these MPI features allow for some optimizations during message matching

no_any_tag / no_any_source assertion

```
if (rank == 0){
 int buffer;
 MPI_Recv(&buffer, 1, MPI_INT,
     MPI_ANY_SOURCE, MPI_ANY_TAG,
     MPI_COMM_WOLRD, MPI_STATUS_IRGNORE);
}else{ // rank == 1
 int buffer;
 MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
     MPI_COMM_WOLRD);
}
```

Listing 7: Standard conform MPI code snipped

- MPI_ANY_TAG and MPI_ANY_SOURCE s to implement non deterministic communication schemes

+ Not using these MPI features allow for some optimizations during message matching

⇒ **Easy to determine if a predefined constant value is used within MPI operations**

Outline

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Evaluated our tool using 48 different small self-written MPI programs.
- Specifically designed to test various cases of $\mathrm{mpi\_allow\_overtaking}$

✓ Minimal impact on compilation time
  - $\mathrm{-ftime\text{-}report}$ reports that our pass uses only 0.3% (0.0014 seconds) of the compilation time for test program ($\approx$ 400 lines of code)

✓ All cases correct for $\mathrm{mpi\_no\_any\_tag}$

✓ All cases correct for $\mathrm{mpi\_no\_any\_source}$
  - simple constant checking
  - no need for extensive evaluation

✓ All cases correct for $\mathrm{mpi\_exact\_length}$
  - more refined implementation desirable
  - with more extensive evaluation

- Most cases correct for $\mathrm{mpi\_allow\_overtaking}$
  - ✓ our tool only suggests using the assertion when it is safe to do so
  - × but it misses some of the cases where one can specify the assertion (see next slides)

Missed cases of allow_overtaking

Interleaved communication and synchronization

```
if (rank == 0){
    MPI_Recv(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD,MPI_STATUS_IGNORE);
    MPI_Ibarrier(MPI_COMM_WORLD, &bar_req);
    MPI_Wait(&bar_req, MPI_STATUS_IGNORE);
    MPI_Recv(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD,MPI_STATUS_IGNORE);
}else{ // rank == 1
    MPI_Ibarrier(MPI_COMM_WORLD, &bar_req);
    MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD);
    MPI_Wait(&bar_req, MPI_STATUS_IGNORE);
    MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD);
}
```

Listing 8: The messages can not overtake in this case

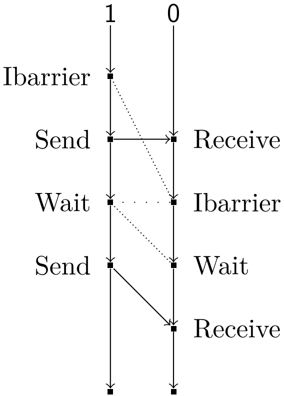


Figure: Illustration of the code snipped

Missed cases of allow_overtaking

Interleaved communication and synchronization

```
if (rank == 0){
    MPI_Recv(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD,MPI_STATUS_IGNORE);
    MPI_Ibarrier(MPI_COMM_WORLD, &bar_req);
    MPI_Wait(&bar_req, MPI_STATUS_IGNORE);
    MPI_Recv(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD,MPI_STATUS_IGNORE);
}else{ // rank == 1
    MPI_Ibarrier(MPI_COMM_WORLD, &bar_req);
    MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD);
    MPI_Wait(&bar_req, MPI_STATUS_IGNORE);
    MPI_Send(&buffer, 1, MPI_INT, 0, TAG,
        MPI_COMM_WOLRD);
}
```

Listing 8: The messages can not overtake in this case

- The first sending operation is "within" the Ibarrier
  - part of the pre-synchronization communication phase
  - part of the post-synchronization communication phase
- Our analysis detects a conflict in the post-synchronization communication phase in this example

Missed cases of allow_overtaking

Ring communication scheme

```
int pre = (rank + 1) % comm_size;
int next = (rank - 1) % comm_size;
// "forward" communication
MPI_Send(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD ,MPI_STATUS_IGNORE);
// "backward" communication
MPI_Send(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD, MPI_STATUS_IGNORE);
```

Listing 9: Ring communication scheme

(pretend that one rank communicates "backward" first to avoid deadlock)

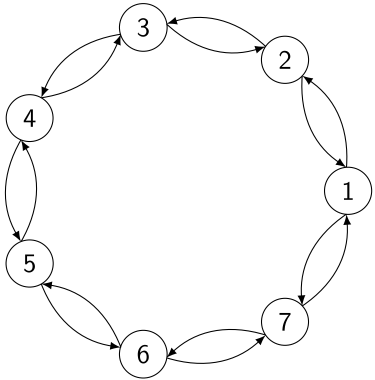


Figure: Ring communication scheme

Missed cases of allow_overtaking

Ring communication scheme

```
int pre = (rank + 1) % comm_size;
int next = (rank - 1) % comm_size;
// "forward" communication
MPI_Send(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, next, TAG,
    MPI_COMM_WOLRD, MPI_STATUS_IGNORE);
// "backward" communication
MPI_Send(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD);
MPI_Recv(&buffer, 1, MPI_INT, pre, TAG,
    MPI_COMM_WOLRD, MPI_STATUS_IGNORE);
```

Listing 10: Ring communication scheme

(pretend that one rank communicates "backward" first to avoid deadlock)

- Static analysis can not prove that pre and next are different
  - if executed with 2 ranks they are same
- Therefore our tool has to assume the sending operations may conflict
- Using different message tags for "forward" and "backward" communication mitigates this problem

Outline

Summary

- Our prototype implemenation shows that detecting if the newly defined communicator info hints hold is possible by static analysis only in many cases

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Our prototype implemenation shows that detecting if the newly defined communicator info hints hold is possible by static analysis only in many cases
- Our analysis is currently limited to the scope of one object file
  - but one can limit the scope of an assertion to one object file by using a duplicate MPI communicator for each object file

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Our prototype implemenation shows that detecting if the newly defined communicator info hints hold is possible by static analysis only in many cases
- Our analysis is currently limited to the scope of one object file
  - but one can limit the scope of an assertion to one object file by using a duplicate MPI communicator for each object file
- We plan to extend our Pass, so that it insert the specification of the assertion if it holds
- We plan to extend our tool to give the programmer guidance on how to change an application so that the assertions hold
- Once MPI implementations exploit the associated performance benefits, we plan an empirical evaluation on the performance gained by specifying the assertions

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Our prototype implemenation shows that detecting if the newly defined communicator info hints hold is possible by static analysis only in many cases
- Our analysis is currently limited to the scope of one object file
  - but one can limit the scope of an assertion to one object file by using a duplicate MPI communicator for each object file
- We plan to extend our Pass, so that it insert the specification of the assertion if it holds
- We plan to extend our tool to give the programmer guidance on how to change an application so that the assertions hold
- Once MPI implementations exploit the associated performance benefits, we plan an empirical evaluation on the performance gained by specifying the assertions
- As we operate on the LLVM IR our approach and implementation is easily transferable to other LLVM input languages
- Our code is available online: https://github.com/tudasc/mach

Scientific Computing

Hessisches Kompetenzzentrum für Hochleistungsrechnen

References

- Message Passing Interface Forum.
  MPI: A Message-Passing Interface Standard 2019 Draft Specification.
  https://www.mpi-forum.org/docs/drafts/mpi-2019-draft-report.pdf, 2019.
- Whit Schonbein, Matthew GF Dosanjh, Ryan E Grant, and Patrick G Bridges.
  Measuring multithreaded message matching misery.
  In *European Conference on Parallel Processing*, pages 480–491. Springer, 2018.