Motivation
Why is runtime analysis in HPC challenging?

- Large amount of resources are used
- State of the program can get very complex → Hard to survey
- Long run duration
Why is runtime analysis in HPC challenging?

- Large amount of resources are used
- State of the program can get very complex → Hard to survey
- Long run duration

- Is there a way to complement dynamic tooling?
Static analysis

- Extensive static analysis of the source code
- Executed in the frontend
- Verify focused aspects of the code

In contrast to 'normal' compiler errors, warnings:
- More computational resources are used
- Better suited for domain specific checks
What are the benefits of static analysis for MPI?

- Analysis without running the program
- Unrelated to runtime resources
- Not affected by the commonness of a sequence at runtime
- Low maintenance effort
Clang Static Analyzer
Clang Static Analyzer

- Framework for static analysis: Core and checkers
- Provides two techniques to base a checker upon:
  - AST-based analysis
  - Path-sensitive analysis
- Descriptive HTML reports
- Extensible
AST-based analysis

```c
void memory(int x) {
    int *i = malloc(sizeof(int));
    if (x < 2) free(i);
    if (x > 0) free(i);
}
```

- Works if a check can verify an invariant locally
- No differentiation of distinct paths
- No assumptions can be made about the range of x

- range of x is unknown
Path-sensitive analysis

```c
void memory(int x) {
    int *i = malloc(sizeof(int));
    if (x < 2) free(i);
    if (x > 0) free(i);
}
```

- Distincts path sequences
- Symbolic execution
- Higher level of abstraction
Symbolic execution

- Symbolic representation of values, memory regions
- Variables are defined by constraints to ranges
- Each node represents a program point and state
- Operations are conceptually transitions between nodes
MPI-Checker
MPI-Checker

- Realised as a Clang Static Analyzer checker
- Hybrid: Provides AST-based and path-sensitive checks
- Can verify C and C++ code
- Checks are MPI implementation independent
Path-sensitive Checks
Path-sensitive checks

- Check aspects of nonblocking communication
- Based on MPI request usage verification
- Request can be in different last user states
  - Unused, used by nonblocking call, used by wait
- Requests are tracked by their symbolic memory region
Double nonblocking

- Nonblocking call using a request that is already in use by a nonblocking call
- Checked when a call is symbolically executed

- Makes it impossible to wait for both nonblocking calls
Unmatched wait

- Checks for waits on requests not used by a nonblocking call

- Request is in an undefined state $\rightarrow$ undefined behavior
Missing wait

- Checks if a nonblocking call is not matched by a wait
- Checked when a symbol goes out of scope

Nonblocking operation might not complete
AST-based Checks
Type mismatch

```
1    int buf;
2    MPI_Send(&buf, *, MPI_DOUBLE, *, *, *);
```

- Buffer type, MPI datatype tag correspondence
Type mismatch

```c
1  int  buf;
2  MPI_Send(&buf, *, MPI_DOUBLE, *, *, *);
```

- Buffer type, MPI datatype tag correspondence
- Clang already has type checking support limited to MPICH
  → MPI-Checker is MPI implementation independent
Type mismatch

```c
1 int buf;
2 MPI_Send(&buf, *, MPI_DOUBLE, *, *, *);
```

- Buffer type, MPI datatype tag correspondence
- Support for all types defined by the MPI 3.1 standard
- Skipped: Custom buffer types, nullpointer constants, custom MPI types, MPI_BYTE, MPI_DATATYPE_NULL
Incorrect buffer referencing

1. `int **buf;`
2. `MPI_Send(buf, *, MPI_INT, *, *, *);`

- MPI functions specify `void *` as their buffer type
- Allows passing pointers not sufficiently dereferenced
- Subroutine of the type mismatch check
Invalid argument type

```c
1 int x = 0;
2 MPI_Send(1.1 + x, *, *, *, *, *);
```

- Check if non-integer types are used for rank, count or tag
- Can handle expressions of arbitrary complexity
- Corresponds to `-Wfloat-conversion`
  - `-Wfloat-conversion` can produce a lot of output
  - `-Wfloat-conversion` is neither included in `-Wall` nor `-Wextra`
- Convenience check
Unmatched point-to-point call

1. `MPI_Send(*, 1, MPI_INT, f() + N + 3 + rank + 1, 0, C);`
2. `MPI_Recv(*, 1, MPI_INT, N + f() + 3 + rank - 1, 0, C, *);`

- Checks for unmatched point-to-point operations
- Names, values must be equal
- Rank needs a specific notation
- Will be changed to a path-sensitive check

Partner a: rank x
Partner b: rank y
Unreachable call

```c
if (rank == 0) {
    MPI_Send(*, 1, MPI_INT, rank + 1, 0, C);
    MPI_Recv(*, 1, MPI_INT, rank + 1, 0, C, *);
}
else if (rank == 1) {
    MPI_Send(*, 1, MPI_INT, rank - 1, 0, C);
    MPI_Recv(*, 1, MPI_INT, rank - 1, 0, C, *);
}
```

- Checks for deadlocks caused by blocking calls
- Based on the same point-to-point matching mechanism
Limitations
Limitations

- No assumption about runtime dependent results can be made → MPI_Waitany or MPI_Waitsome are not taken into account
- Heap allocated MPI_Request variables are not taken into account
- Analysis is limited to the scope of a translation unit
Evaluation
## Evaluation

- AMG2013 ~75KLOC, 10x
- CombBLAS ~40KLOC, 2x
- OpenFFT ~5KLOC, 4x

- No false positives but the likeliness of appearance differs
- Point-to-point checks were excluded
# AMG2013 - Report overview

<table>
<thead>
<tr>
<th>Bug Group</th>
<th>Bug Type</th>
<th>Function/Method</th>
<th>Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI Error</td>
<td>Double nonblocking</td>
<td>hypreDataExchangeList</td>
<td>23</td>
</tr>
<tr>
<td>MPI Error</td>
<td>Double nonblocking</td>
<td>hypreDataExchangeList</td>
<td>23</td>
</tr>
<tr>
<td>MPI Error</td>
<td>Incorrect buffer referencing</td>
<td>hypre_BoxManAssemble</td>
<td>1</td>
</tr>
<tr>
<td>MPI Error</td>
<td>Missing wait</td>
<td>hypreDataExchangeList</td>
<td>29</td>
</tr>
<tr>
<td>MPI Error</td>
<td>Type mismatch</td>
<td>hypre_CSRMatrixToParCSRMatrix</td>
<td>1</td>
</tr>
<tr>
<td>MPI Error</td>
<td>Unmatched wait</td>
<td>hypreDataExchangeList</td>
<td>26</td>
</tr>
</tbody>
</table>
MPI_Request *term_requests, term_request1, request_parent;

if (!response_obj_size) response_obj_size = sizeof(int);

Assuming 'response_obj_size' is not equal to 0

Taking false branch

if (!contact_obj_size) contact_obj_size = sizeof(int);

Assuming 'contact_obj_size' is not equal to 0

Taking false branch
AMG2013 - Detail report - Missing wait

```c
MPI_Irecv(NULL, 0, MPI_INT, tree.parent_id, term_tag, comm,
&term_request1);
```

← Request is previously used by nonblocking call here. →

← Request 'term_request1' has no matching wait. →
AMG2013 - Detail report - Type mismatch

MPI_Bcast(&global_data[3], global_size-3, MPI_INT, 0, comm);

Buffer type 'long long' and specified MPI type 'MPI_INT' do not match.
Future Work
Future work

- Merge MPI-Checker into Clang
- Detect race condition on buffer between nonblocking call and wait
- Path-sensitive point-to-point matching
- Possibility to type match custom types
- Analysis for a given process count
- ...
  → Adding new checks will now be a lot easier
<table>
<thead>
<tr>
<th>Motivation</th>
<th>Clang Static Analyzer</th>
<th>MPI-Checker</th>
<th>Limitations</th>
<th>Evaluation</th>
<th>Future Work</th>
</tr>
</thead>
</table>

Current State
Current state

- GitHub: https://github.com/0ax1/MPI-Checker
- Range of checks
- Limitations
- Examples
- Planned: Evaluation
## Acknowledgments
Acknowledgments

- Hal Finkel
- Anna Zaks
- Dmitri Gribenko
- Devin Coughlin
- Jeff Hammond
+ Clang mailing list
Questions?