Overview

In the context of MPI (Message Passing Interface, [1]), our talk presents a long-standing collaboration between RWTH Aachen and TU Darmstadt to further extend the MPI correctness checker tool MUST [2] with memory allocation checking capabilities in the context of MPI (communication) calls for C/C++ target programs. To that end, we developed the LLVM compiler plugin TypeART [3]. Both tools are open source and available under the BSD 3-clause license.

In general, MUST’s correctness checking includes errors that already manifest — segmentation faults or incorrect results — and many errors that are invisible to the developer or only manifest with certain HPC systems and MPI libraries. MUST works by intercepting MPI calls of a target application at runtime, allowing for bookkeeping of the current program state. Thus, MUST can cope with the complex MPI semantics of, e.g., (a) collectives, (b) wildcards, or (c) datatypes.

As a consequence of relying on intercepting MPI calls, though, MUST is unaware of the effective type of the allocated void* buffers used for the low-level MPI API. To that end, TypeART was developed to track memory (de-) allocation relevant to MPI communication. TypeART instruments heap, stack and global variable allocations with a callback to our runtime. The callback consists of (a) the memory address, (b) the type-layout information of the allocation (built-ins, user-defined structs etc.) and (c) number of elements. Thus, with TypeART, MUST can check for type compatibility between the type-less MPI communication buffer and the declared MPI datatype.

Consider the MPI function MPI_Send(const void* buffer, int count, MPI_Datatype datatype,...). Without TypeART, MUST cannot check if the buffer argument is compatible with the declared MPI_Datatype and if the count argument exceeds the buffer allocation size. Consider the following code snippet:

```c
// Somewhere in a code, TypeART instruments/tracks this allocation (memory address, type and size):
float* array = (float*) malloc(length*sizeof(float));

// Sometime, MUST intercepts this MPI call, and asks TypeARTs runtime library for type information:
MPI_Send((void*)array, length, MPI_DOUBLE, ...)
```

In the above example, the first check already fails, as there is a mismatch between the user-allocated buffer handle and the specified MPI datatype. Our tools also handle derived datatypes [1, Sec. 5] with complex underlying C/C++ data structures.

Structure of the talk

1. Give a motivating small MPI program snippet that is full of errors (and can be checked by MUST & TypeART)
2. Show (HTML) error reports based on the snippet produced by MUST
3. Introduce MUSTs & TypeARTs architecture and usage
4. Focus talking point: derived datatype. (If time permits)
5. Evaluation of overhead (MUST and the instrumentation with TypeART) based on well-known HPC mini apps

References

[2] https://itc.rwth-aachen.de/must/
[3] https://github.com/tudasc/TypeART