MPI Tools the Easy Way

Bengisu Elis[1] , Martin Ruefenacht[2], Olga Pearce[3] , Kathryn Mohror[3] ,Martin Schulz[1] , Anthony Skjellum[2]

[1] Technical University of Munich, [2] The University of Tennessee at Chattanooga,
[3] Lawrence Livermore National Laboratory

Motivation

1. Different languages have different strengths and this work is a stepping stone towards support for other languages in order to implement MPI tools.
2. C/C++ and Fortran are advantageous for high performance, but software development in these languages is challenging.
3. Python is straightforward, portable, and it doesn't require complicated environments.
4. Python especially enables easy development of:
   - Interactive debugging for MPI applications
   - MPI call counting
   - Runtime MPI argument checking as a debugger

Mechanism

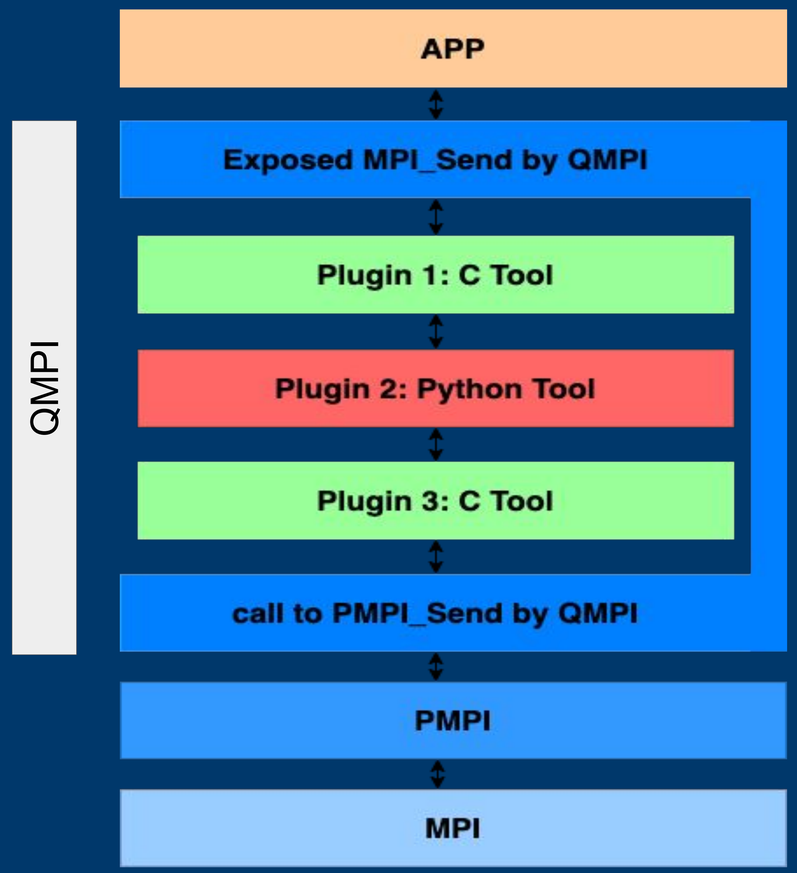
Setup workflow:

1. QMPI-mock prototype will be linked with the application instead of MPI library
2. List all tool paths in the TOOLS environment variable
3. The application can be executed the same way as a regular MPI application:
   mpiexec -n 5 ./application

Runtime workflow:

1. QMPI-mock intercepts the MPI_Init call from the application and sets up function pointer tables to establish execution order of tool routines.
2. QMPI-mock calls the python interpreter to discover the references for routines from python tools and initialize the python environment. The references are added to the function pointer tables.
3. Tools request and execute function pointers to routines which belong to the tool via QMPI-mock provided services.

We use **Python** to develop **MPI Tools** through **QMPI-mock**.



QMPI | Contact | This Poster

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-807339).

Discussion

- Python provides a large standard library which C/C++ and Fortran do not.
- Developers do not need to know the internals of the function resolution.
- Using QMPI-mock for interfacing requires changes in the MPI tool implementation, hence in the MPI standard.
- Exact workflow and its improvement are future-work.

Results

- A Python tool can be developed by a user easily and quickly.
- Python allows overloading of functions without having to add boilerplate code
- Python enables usage of same handler function for multiple MPI functions.
- Python tool is interpreted whereas C tool must be compiled and linked after every change in the code.
- **However;** a certain overhead is expected due to invocation of the CPython interpreter and the conversion to Python types.

Python tool example code :

```
from qmpi import register_handler, Invocation

@register_handler("MPI_Init")
def python_MPI_Init (invocation):
    # handle the invocation of MPI_init

    return invocation.descend()
```

Equivalent C tool:

```
#include qmpi.mock.h

int  c_MPI_Init (int *argc, char ***argv, int i , vector* v )
{
    // handle the invocation of MPI_init

    void* function_ptr ;
    Int ret;
    QMPI_Table_query ( _MPI_Init, function_ptr );
     ret = exec_func( function_ptr , _MPI_Init ,  argc, argv);
    return ret;
}
```