Verification of MPI Programs Using Session Types

Kohei Honda[1], Eduardo R.B. Marques[2], Francisco Martins[2],
Nicholas Ng[3], Vasco T. Vasconcelos[2], and Nobuko Yoshida[3]

[1] Queen Mary & West Field College, University of London, UK
[2] LaSIGE, Faculty of Sciences, University of Lisbon, PT
[3] Imperial College London, UK

Developing safe, concurrent (and parallel) software systems is a hard task in multiple aspects, particularly the sharing of information and the synchronization among multiple participants of the system. In the message passing paradigm, this is achieved by sending and receiving messages among different participants, raising a number of verification problems. For instance, exchanging messages in a wrong order may prevent the system from progressing, causing a deadlock. MPI is the most commonly used protocol for high-performance, message-based parallel programs, and the need for formal verification approaches is well acknowledged by much recent work (e.g., see [1]).

Our proposal for verification of MPI programs is based on session types [3]. The methodology considers the specification of a global interaction protocol among multiple participants, from which we can derive an endpoint protocol for each individual participant, e.g., as in Scribble [2]. A well-formed protocol can be verified in polynomial time and ensures type safety, communication safety, and deadlock freedom [4]. The idea is that we can ensure these properties for an MPI program by verifying conformance of the program against a given session type specification. This contrasts with other state-of-the-art methodologies considered for MPI, like model checking or symbolic execution [6], that require program-level analysis for all properties of interest, and inherently lead to a state-explosion problem as the number of participants grows.

Session type

```
process r  :
 r in {0, ..., P-1},
 N > 0
= loop {
 float[N] to   (r+1) % P
 float[N] from (P+r-1) % P
 float    allreduce
}
```

MPI fragment

```
float err, localErr, sbuf[N], rbuf[N];
int r, P;
MPI_Comm_rank(MPI_COMM_WORLD, &r);
MPI_Comm_size(MPI_COMM_WORLD, &P);
...
for (i=0; i < MAX_ITER && err > MAX_ERROR; i++) {
 MPI_Sendrecv(sbuf, N, MPI_FLOAT, (r+1) % P, 0,
              rbuf, N, MPI_FLOAT, (P+r-1) % P, 0,
              MPI_COMM_WORLD, &status);
 // computation
 ...
 MPI_Allreduce(&localErr, &err, 1, MPI_FLOAT,
               MPI_MAX, MPI_COMM_WORLD);
}
```

To illustrate our proposal we sketch a ring pattern that can be found in many MPI programs, e.g., n-body pipeline computations, shown above. We depict a pseudo-session type specification (left) and a corresponding MPI program fragment (right). The session type specifies that in every turn each participant `r` should send a float array of size `N` to its right neighbor and receive another array of the same size from its left neighbor. Then, after some local computation involving the received data, all participants perform a collective reduction (using `MPI_Allreduce`).

J.L. Träff, S. Benkner, and J. Dongarra (Eds.): EuroMPI 2012, LNCS 7490, pp. 291–293, 2012.

We have identified two key challenges. The first is to refine session type abstractions to capture the general traits of MPI programs, e.g., rank-based communication, collective operations, typical communication patterns (e.g., ring, mesh), and other MPI operations that may correspond to multiple steps in the protocol (like `MPI_Sendrecv` in the example). Other features impose additional complexity, such as nondeterministic operations (e.g., wildcard receives) or the possible choice/coexistence between blocking and nonblocking operations (e.g., an `MPI_Send` operation can be matched by a `MPI_Irecv`/`MPI_Wait` operation pair). Important work such as dependent-types or parameterized multiparty session types [7] can provide insights on these topics.

Session types have already been used to describe and verify parallel programs, e.g., Session C [5]. The proposals so far, however, require that programs are specified using a session type-specific programming abstraction and provide no support for common traits of message-based parallel programs such as collective operations. In contrast, we propose checking the conformance of standard MPI programs against session types. This second challenge is far from trivial. In essence, we need to determine a sound correspondence between a session type specification and the control flow graph of a program *for any* process. The communication flow is dependent on the numerical rank of each process, i.e., for any `r` in the example the endpoint type must be matched against the concrete control flow of the MPI program when executed for rank `r`. Moreover, a control flow synchrony needs to be established between processes. In the example we would need to infer that the same number of loop iterations is executed for all ranks, based on the assertion that `err` and `i` always have the same value in all processes per each iteration (note that `err` results from `MPI_Allreduce`). Beyond this simple example, other MPI programs easily make this type of assertions more complex to infer, e.g., manager-worker programs which combine such a parameterization with distinct branches and communication operations for different process groups.

Acknowledgements. This work is partially supported by EPRSC funds EP/G015635/01 and EP/G015481/01, Fundação para a Ciência e Tecnologia funds PTDC/EIA-CCO/122547/2010, and the Ocean Observatories Initiative.

References

1. Gopalakrishnan, G., Kirby, R.M., Siegel, S., Thakur, R., Gropp, W., Lusk, E., De Supinski, B.R., Schulz, M., Bronevetsky, G.: Formal analysis of MPI-based parallel programs. Communications ACM 54(12), 82–91 (2011)
2. Honda, K., Mukhamedov, A., Brown, G., Chen, T.-C., Yoshida, N.: Scribbling Interactions with a Formal Foundation. In: Natarajan, R., Ojo, A. (eds.) ICDCIT 2011. LNCS, vol. 6536, pp. 55–75. Springer, Heidelberg (2011)
3. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Discipline for Structured Communication-Based Programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
4. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL, pp. 273–284. ACM (2008)

5. Ng, N., Yoshida, N., Honda, K.: Multiparty Session C: Safe Parallel Programming with Message Optimisation. In: Furia, C.A., Nanz, S. (eds.) TOOLS 2012. LNCS, vol. 7304, pp. 202–218. Springer, Heidelberg (2012)
6. Siegel, S., Mironova, A., Avrunin, G., Clarke, L.: Combining symbolic execution with model checking to verify parallel numerical programs. ACM TOSEM 17(2), 1–34 (2008)
7. Yoshida, N., Deniélou, P., Bejleri, A., Hu, R.: Parameterised Multiparty Session Types. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 128–145. Springer, Heidelberg (2010)