Step 3: Safety Properties Verified

TASS can verify a variety of safety properties in a simple MPI program. These include:

• Absence of arithmetic or potential deadlocks
• Absence of buffer overflows from process arithmetic, array indexing, etc.
• Absence of invalid undiagnosed variables
• Absence of division by zero
• Absence of memory leaks
• Proper use of malloc and free
• Absence of assertion violations
• Type and size of a message received with MPI_Irecv are compatible with the receive buffer type

Different ways.

The TASS front end takes an integer n and a CMPB program, and constructs an abstract model of the program with n processes. Procedures, structs, (multidimensional) arrays, heap-allocated data, pointers, and pointer arithmetic are all represented in this TASS model. The model is then explored using symbolic execution and explicit state space enumeration. A number of techniques are used to reduce the time and memory consumed. A variety of realistic MPI programs have been analyzed, including Jacobi iteration and manager-worker type programs, and some subtle defects have been discovered. TASS is written in Java and is available from http://vsl.cis.udel.edu/tass under the GNU Public License.

4. Comparative Symbolic Execution

TASS pragmas can identify output as well as input variables. These annotations specify an input set X and an output set Y. Thus the program can be seen as a function f: X→Y. When two programs have a similar input model constant (X1, X2, ...), TASS can then check that they compute the same function. In TASS, the front end of a program is transformed to the specific function and the second as the implementation.

The TASS front end takes an integer n and a CMPB program, and constructs an abstract model of the program with n processes. Procedures, structs, (multidimensional) arrays, heap-allocated data, pointers, and pointer arithmetic are all represented in this TASS model. The model is then explored using symbolic execution and explicit state space enumeration. A number of techniques are used to reduce the time and memory consumed. A variety of realistic MPI programs have been analyzed, including Jacobi iteration and manager-worker type programs, and some subtle defects have been discovered. TASS is written in Java and is available from http://vsl.cis.udel.edu/tass under the GNU Public License.

5. States

A symbolic state assigns a symbolic expression to each variable in the program, including pc: it represents a set of concrete states: any assignment of concrete values to symbolic constants that satisfies pc results in a concrete state in the set.

As above, the program above the sum of the elements of an array in the two different ways. TASS will verify that for all array lengths up to the green bound B and any possible real number array entries, the result will be the same for both programs using comparative symbolic execution [4]. A composite model is constructed in which the two programs run concurrently. The same symbolic constants are used to initialize the input variables. In this case, the properties on the two programs and a single path condition variable is used. At each terminal state, an assertion comparing the output variables in the two programs is checked. If the assertion holds on all executions, the programs are functionally equivalent.

6. Collective Assertions

TASS provides a powerful generalization of standard assertions called collective assertions [5]. Collective assertions allow for specifying relations among variables in different processes. They are analogous to MPI’s collective operations, but do not add any synchronization or change the semantics of the program in any way.

A collective assertion is not simply an assertion on the global state of the program. To see why, consider a block-distributed 1D diffusion solver with MPI processes. In order for processes to change the state of the grid to the value of the value of their neighbor’s leftmost cell and left neighbor’s rightmost cell. After grid cells are exchanged, it is desirable to assert that they are correct. Each process will repeatedly exchange ghost cells, check an assertion, and update cells. There are two views of this assertion. The first is “my ghost cells are correct.” Below we show one possible execution.

7. Performance

The graphs describe resource use by TASS when analyzing a variety of C programs. In each case the left y-axis is in seconds.

8. Future Work

Several enhancements to TASS are planned. First, support for more C libraries is in progress. In particular, the C math, float, and stdio libraries are short term targets. Additionally, we plan to expand the MPI support to handle non-blocking MPI calls. Work is underway to add support for verifying the correctness of the MPI routine. Current support for C routines such as C++ and Fortran is also in progress. Finally, we plan to support other high-performance computing APIs such as OpenMP, OpenCL, and CUDA.

9. References