Scalable Parallel Debugging via Loop-Aware Progress Dependence Analysis

Subrata Mitra[1], Ignacio Laguna[2], Dong H. Ahn[2], Todd Gamblin[2], Martin Schulz[2], Saurabh Bagchi[1]

[1]Purdue University, West Lafayette, USA [2] Lawrence Livermore National Laboratory, USA

E-mail: {mitra4,sbagchi}@purdue.edu, {lagunaperalt1, ahn1, tgamblin,schulzm}@llnl.gov

Introduction

For many bugs in a scientific application—where parallel tasks progress forward in a coordinated fashion—finding tasks that progressed the least can significantly reduce the time to isolate the root cause. We present a novel run-time technique, the loop-aware progress-dependence analysis, which can improve the accuracy of identifying the least-progressed (LP) task(s). Our technique extends an existing analysis (AutomaDeD) to detect LP task(s) even when the error arises on code with complex loop structures.

Case Study

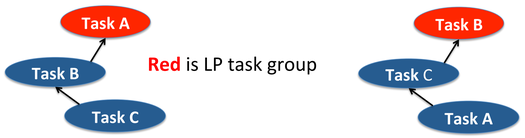
A few MPI tasks in Group B stuck due to a thread-level deadlock while others stop in a wait chain because their progress depends on Group B:

```
10  while (1)
    {
            . . .
            /*Global synch*/
27          MPI_Barrier(MPI_COMM_WORLD);
            . . .
42          consumer();
             . . .
    }
    int consumer ( ) {
          do
          {
            // consume a number produced by producer thread
47          sendbuf = pop_a_random_num( );
            } while (sendbuf == -1);

          //Receiving a random number from the neighbor
          MPI_Irecv( … );
          // Sending a random number to the neighbor
          MPI_Isend( … );
          . . .
68        MPI_Waitall( … );

          . . .
    }
```

Task Group A — Task Group B — Task Group C

- ❖ Identification of LP tasks can systematically localize the origin of such errors.
- ❖ AutomaDeD uses transition probabilities to compute a progress dependency graph (PDG).
- ❖ However, if an error occurs on code with complex loop structures, determining accurate progress dependencies with probabilities alone is becoming difficult.

What is correct the PDG for the above code?

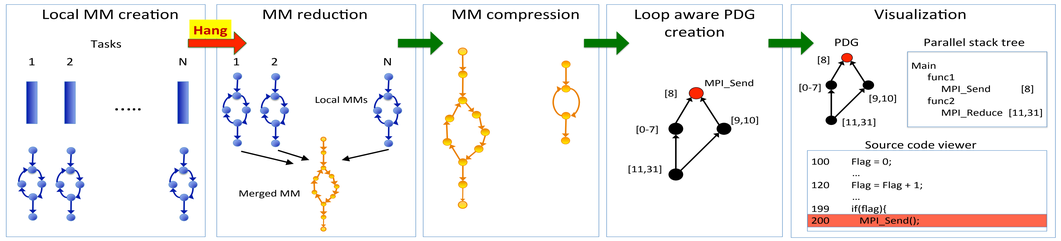


- ✓ The notion of progress dependence of MPI tasks must consider iteration counts of the relevant loops.

Contributions

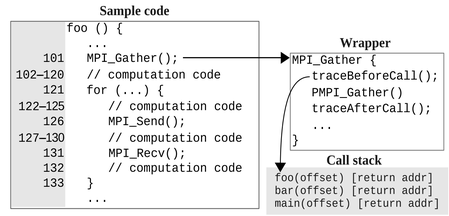
- Loop-aware Markov model that approximates the behavior of a task in loops
- Scalable merging and reduction of these models to represent a global view
- Scalable—yet highly accurate—loop-aware progress-dependency analysis
- Scalable visualization of loop-aware progress-dependency information
- Preliminary evaluations that show better analysis accuracy compared to the baseline.

Design



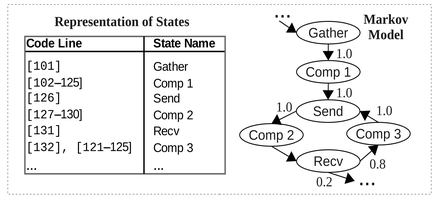
1) Probabilistic model creation:

- Markov model (MM) to capture MPI control flow for each task
- In addition to transition probabilities, edges are augmented with transition counts



2) Generating global view:

- Each task has partial information
- Binomial tree based reduction of MMs
- Scalable aggregation of edge information



3) Compression:

- Reduce the search space by eliminating or combining redundant information

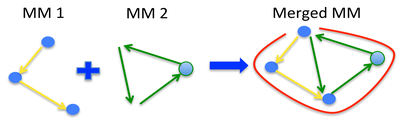
4) Loop aware PDG analysis:

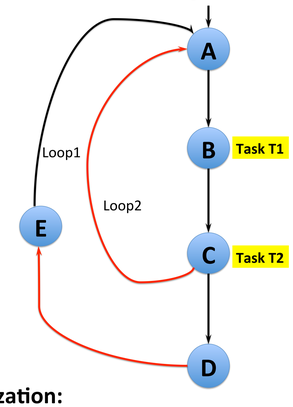
- Divide tasks into small sets of equivalent classes
- Compare progress and determine dependency between two task groups

❖ Current state-of-the-art: probabilistic dependency analysis:

- ✓ Based on forward and backward path transition probabilities
- ✓ L imitations when no single dominant probability exists

MM merging may introduce new loops





Dependency inside loop(s):

Infers dependency by comparing iteration counts of loops:

- **Dynamic analysis:** iteration counts inferred from transitions
- **Characteristic Edge:** an edge may be part of several loops; characteristic edge uniquely represents the iteration count of that loop
- **Lexicographic ordering**
- **Comparing distances:** In case of a tie, use distance from loop entry point to compare progress
- **LP task(s)**: does not depend on any other tasks

5) Visualization:

- Present PDG with associated aggregate stack-traces for a selected set of tasks groups and source code listing

Preliminary Results

We tested our tool with cases where the state-of-the-art tool (AutomaDeD) gave incorrect results. We also did a fault injection study on AMG by injecting infinite loops at 100 random locations. Our technique significantly improved the accuracy.

Bug type	Application type	Baseline	Loop Aware
Deadlock	Producer-Consumer	Incorrect PDG/LP-task (Infinite loop)	Correct PDG
Hang	Poisson Solver	No PDG (Infinite loop)	Correct PDG
Hang	AMG	82% accuracy	100% accuracy

Benchmark	Slowdown	Memory overhead
AMG	1.9	2.1
LAMMPS	2.2	1.6
LULESH	1.2	6.5

Further Study plan

❖ Performance analysis:

- Runtime overhead for model creation
- Analysis scale with thousands of tasks
- Complex code with large MM model

❖ Accuracy and precision analysis:

- Random fault injection experiments
- Various fault types
- More real-world case studies

Conclusion and Future work

- An ability to relate tasks on their relative progress is a powerful root-cause localization
- Our loop-aware progress-dependence analysis extends AutomaDeD to infer progress dependencies with high accuracy without compromising performance and scalability
- We plan to extend our technique to support more complex loops and further evaluate its utility and performance.

Acknowledgement

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-641599).