Dynamic Verification of Hybrid Programs*

Wei-Fan Chiang[1], Grzegorz Szubzda[1],
Ganesh Gopalakrishnan[1], and Rajeev Thakur[2]

[1] School of Computing, Univ. of Utah, Salt Lake City, UT 84112, USA
[2] Math. and Comp. Sci. Div., Argonne Nat. Lab., Argonne, IL 60439, USA

Overview

Hybrid (mixed MPI/thread) programs are extremely important for efficiently programming future HPC systems. In this paper, we report our experience adapting ISP [3,4,5], our dynamic verifier for MPI programs, to verify a large hybrid MPI/Pthread program called Eddy Murphi [1]. ISP is a stateless model checker that works by replaying schedules leading up to previously recorded non-deterministic selection points, and pursuing new behaviors out of these points. The main difficulty we faced was the inability to *deterministically replay* up to these selection points because ISP instruments only the MPI calls issued by an application, whereas thread level scheduling non-determinism may change the course of execution. Instrumenting both MPI and Pthreads API calls requires an invasive modification of ISP which was not favored. The novelty of our solution is to determinize thread schedules using a record/replay daemon and demonstrating that this approach works on a realistic hybrid application: the Eddy Murphi model checker.

Verification Challenge

Figure 1 illustrates the architecture of Eddy Murphi, a parallelized and distributed model checker. It essentially implements a BFS approach algorithm to explore the state space. Each process of Eddy Murphi (the node shown in Figure 1) consists of the worker thread and the communicator thread (CT). The communicator thread issues MPI sends and MPI wildcard receives.

If we are to successfully verify Eddy Murphi using ISP, we must have ISP explore the space of non-deterministic receives of the communicator threads. ISP must not be "confused" by the Pthread schedule changes that may vary the order in which the worker and communicator threads obtain the mutex lock that guards common data structures to these threads.

Determining Solution

ISP's operation is as follows. It collects MPI calls from all processes and waits for the processes to reach their *fence points*. Once all processes have reached fences, ISP chooses a sender process in the set of potential matching senders

* Supported in part by Microsoft, NSF CNS-0935858, CCF-0903408, and DOE ASCR DE-AC02-06CH11357.

R. Keller et al. (Eds.): EuroMPI 2010, LNCS 6305, pp. 298–301, 2010.

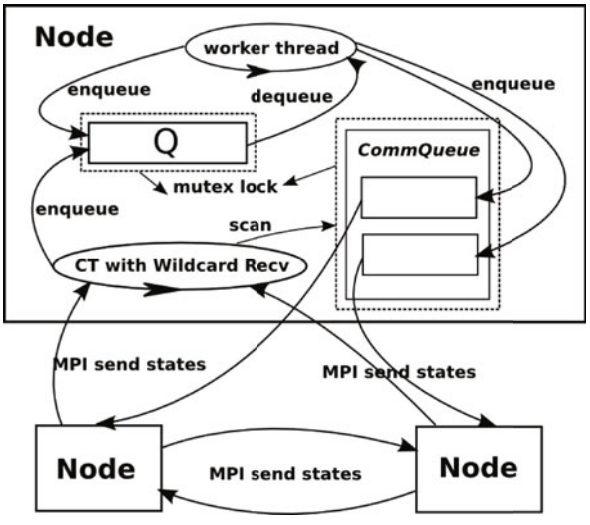

Fig. 1. The Architecture of Eddy Murphi

to a wildcard receive and rewrites the wildcard receive into a specific receive matching this sender, thus determinizing the *MPI schedule* from that point. We call such a step a determinized receive (*DR event*) event.

Figure 2 illustrates how we augment ISP with a daemon that helps record a previous Pthread schedule (of Pthread mutex calls, *etc.*) up to a DR event into a log file and enforces the recorded schedule when we replay up to this DR event. Essentially, the daemon sends positive acknowledgements (ACKs) to threads calling Pthread routines in an order matching the recorded order, and sends negative acknowledgements (NACKs) to threads calling in a non-matching order. The NACKed threads have to re-issue their Pthread calls or be blocked on the calls.

For the very first schedule exhibited, our daemon is in the record mode, recording the entire schedule. When replaying the schedule, the daemon stays in the replay mode till it sees ISP pursue a new behavior (a different DR event is pursued by ISP). The daemon then switches to record mode, extending the partially replayed schedule with a new record sequence. This concatenated sequence forms the 'seed' for the next execution. Such a record/replay mechanism is easily implemented by keeping only two log files: the previous one and the current one.

Assumptions (satisfied by Eddy Murphi):

- All read/writes are protected by mutual exclusion locks.
- The MPI threading level is `MPI_THREAD_FUNNELED`.
- Processes communicate only through MPI calls.
- Other API calls (besides MPI and Pthreads) are not allowed.
- The inputs provided by our test harness are deterministic.

Related Works: The idea of our "record/replay" mechanism is inspired by ODR, output deterministic replay [2]. Our "record/replay" method is similar

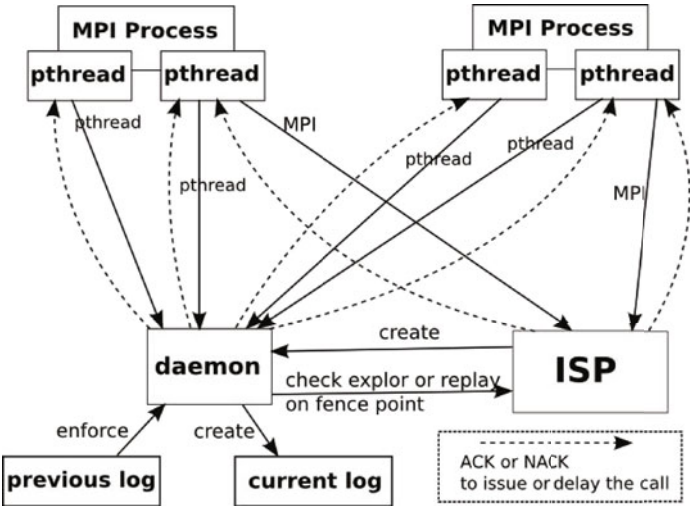

Fig. 2. ISP-Daemon System

to the SI-DRI recording approach to recording the lock order used in ODR. ODR gathers the schedule trace, input trace, and read trace from the original execution and generates symbolic path constraints. These are then solved using a constraint solver to enforce previous schedules. Our mechanisms are much simpler in comparison.

Findings and Concluding Remarks

We chose the *n_peterson* model as the test input of Eddy Murphi and set the depth bound of it to control the scale of exploration. Table 1 presents our experiment results. The *original isp* denotes the old version of ISP which cannot handle thread non-determinism (it crashes when threads change their schedules). The *isp-daemon* denotes the new version of ISP which can enforce Pthread schedules. The column *ISP version & configuration* denotes the version of ISP we used, the

Table 1. Experiment on n_peterson Model

ISP version & configuration	interleaving explored	min./max./ave. DR events
original isp / p3 / d3	11	112 / 112 / 112
original isp / p3 / d5	fail on 2ed	133 / 133 / 133
original isp / p4 / d3	fail on 4th	145 / 149 / 146
isp-daemon / p3 / d3	11	112 / 112 / 112
isp-daemon / p3 / d5	61	133 / 133 / 133
isp-daemon / p3 / d10	over 1500	179 / 179 / 179
isp-daemon / p3 / d20	over 1600	723 / 765 / 727
isp-daemon / p4 / d3	6	141 / 145 / 143
isp-daemon / p4 / d5	over 1097	174 / 174 / 174
isp-daemon / p4 / d10	over 2000	300 / 304 / 303
isp-daemon / p4 / d20	over 2400	898 / 898 / 898

number of processes, and the depth bound. For instance, "original isp / p3 / d5" means the result of running Eddy Murphi on the old version ISP with three processes created and with a 5-level depth bound BFS. The column *interleaving explored* denotes the number of interleavings explored by ISP while verifying Eddy Murphi. The column *min./max./ave. DR events* denotes the minimum, maximum, and average number of DR events we encountered in one execution.

The main limitation of our method is that it does not guarantee coverage of the Pthread non-determinism space. It covers only the MPI non-determinism space for particular determinizations of the Pthread schedule space. We briefly explored trying to iterate through the Pthread schedule space through methods such as preemption bounded search, and re-running ISP for each such altered Pthread schedule. Such an approach will result in an extremely large schedule space to cover – equaling the product of the Pthread and MPI schedule spaces. A better approach may be to conduct a random-walk across the Pthread and MPI schedule spaces. Our main conclusion is that unless hybrid programming is approached with discipline, building a tractable verification approach becomes nearly impossible. Our future work will examine how to make ISP capable of dynamically verifying more types of hybrid programs such as MPI-OpenMP, MPI-CUDA, *etc.*

From a practical point of view, it is important to cover mixed MPI/OpenMP programs. However, this opens up a number of challenges. First, it would be necessary for *OpenMP implementors to expose the underlying OpenMP scheduling points.* Without this, it would be impossible to guarantee any sort of coverage. Since OpenMP implementations differ significantly from each other, ideally one would standardize such an API pertaining to scheduling so that hybrid dynamic verifiers can resort to a single standardized solution in determinizing the OpenMP behaviour while exploring MPI behaviours.

Even with such help, we do need a well articulated set of programming practices around which to build dynamic verifiers for hybrid programs. Without such programming practices, the product schedule spaces of the individual concurrency models will be so huge that it cannot be covered within reasonable amounts of time.

References

1. Eddy murphi. distribution, `http://www.cs.utah.edu/formal_verification/mediawiki/index.php/Eddy_Murphi`
2. Altekar, G., Stoica, I.: ODR: Output-deterministic replay for multicore debugging. In: 22nd symposium on Operating Systems Principles (SOSP), pp. 193–206 (2009)
3. Vakkalanka, S.: Efficient Dynamic Verification Algorithms for MPI Applications. PhD thesis (2010), `http://www.cs.utah.edu/Theses`
4. Vakkalanka, S., Gopalakrishnan, G., Kirby, R.M.: Dynamic Verification of MPI Programs with Reductions in Presence of Split Operations and Relaxed Orderings. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 66–79. Springer, Heidelberg (2008)
5. Vo, A., Vakkalanka, S., DeLisi, M., Gopalakrishnan, G., Kirby, R.M., Thakur, R.: Formal verification of practical mpi programs. In: PPoPP, pp. 261–269 (2009)