ABSTRACT

Formal dynamic verification complements MPI program testing by detecting hard-to-find concurrency bugs. These bugs are elusive and difficult to reproduce and track down using conventional methods due to the difficulty in determining the state of a program when the bug occurs. In previous work, we described our dynamic verifier called In-situ Partial Order (ISP [5]) that can efficiently search the execution space of an MPI program while detecting important classes of bugs. ISP does this by executing processes under the control of its own scheduler. One major limitation of ISP, when used by itself, is the lack of a powerful and widely usable graphical front-end. We now present a new tool called Graphical Explorer of MPI Programs (GEM [2]) that overcomes this limitation. GEM is designed as an Eclipse plug-in within the Parallel Tools Platform (PTP [1]), a rapidly evolving tool integration framework for parallel program development and analysis. GEM serves to bring ISP within reach of a wide array of programmers with its original release as part of PTP version 3.0 in December, 2009. Our poster illustrates the motivation for GEM, an overview of its features and our future plans.

Categories and Subject Descriptors

D.0 [Software]: General; I.6.4 [Simulation and Modeling]: Model Validation and Analysis

General Terms

Design, Theory, Verification

Keywords

Dynamic Verification, Graphical User Interfaces, Dynamic Interleaving Reduction, Message Passing, MPI, Multi-core, Eclipse Parallel Tools Platform.

1. INTRODUCTION

The Message Passing Interface (MPI [3]) library continues to enjoy a dominant position in HPC, and has been ported to run on virtually every parallel machine available today. Given the extensive presence of MPI, it is imperative that highly effective, debugging tools with intuitive graphical front-ends be created for MPI programs. These debugging tools must strive to meet certain goals: (i) reliably localize errors and display these errors intuitively, (ii) display only relevant information about a particular execution, avoiding redundancy, (iii) depict items of interest so that users gather deeper insights about their programs and the APIs/libraries they use, (iv) help locate root causes of bugs found, (v) provide facilities to step through and visualize process executions. These goals are even more important to meet for concurrent program debuggers, as concurrent program executions are often less intuitive than sequential program executions.

Today, there is an impressive array of tools available for debugging MPI programs, unfortunately, these tools only provide ad hoc techniques for process interleaving (schedule) generation, and as a result, many interleavings are not considered. In practice, these omitted interleavings often harbor elusive bugs. Considering all interleavings is not an option, as even a simple MPI program with five processes, each making five calls has potentially 10 billion interleavings (schedules). It can be easily seen though, that large groups of interleavings are totally equivalent. ISP has the ability to determine relevant interleavings (possible schedules with runtime behavior different from those already observed), and the ability to enforce interleavings.

While ISP was a significant step forward in the arena of dynamic verification tools for MPI, its usage was hindered by the absence of a widely usable and intuitive graphical user interface. This poster illustrates our contribution in this regard through a new tool called Graphical Explorer of MPI Programs (GEM).

2. GEM FEATURES

GEM makes ISP truly a push-button dynamic formal verification tool for MPI C/C++ codes with its tight integration within the Eclipse IDE (Figure 1). GEM offers support for both Eclipse C/C++ Development Tooling (CDT) Managed Build and Makefile projects.
The Analyzer view is the primary Eclipse view available through GEM. This view gives a user-friendly debugger look and feel and has the ability to present verification results as if the matches happened according to program order as well as the ability to view instruction execution by internal execution order (the order in which ISP’s scheduler issues calls to the MPI runtime). Other facilities are provided by GEM that allow a developer to quickly localize the source of deadlocks, MPI resource leaks and local assertion violations (Figure 2). A developer can then visit the offending lines of code within the Eclipse editor with a single mouse click.

Figure 2: GEM’s Resource Leak Browser

With the integration of GEM into PTP, an MPI developer can subsequently write, debug, formally verify and launch their MPI code on a cluster, all from within the Eclipse IDE. PTP also offers a scalable, parallel debugger.

3. CASE STUDY: VERIFYING PARMETIS

It was our goal not only to make GEM useful as an academic tool, but also that it be viable for larger scale codes. GEM proved highly capable with all of our smaller projects and most medium scale Makefile projects. This included a Life simulation verified with up to 20 processes in under three minutes. Of particular interest though, as to the real viability and scalability of the GEM/ISP combination, is the ParMETIS case study.

ParMETIS 3.1 ([4]) is a widely used parallel graph partitioning and sparse matrix ordering library. Verifying ParMETIS makes for an excellent study due to the inherent complexities involved with verifying a project of such size. Some ParMETIS routines have more than 12,000 lines of code between themselves and their helper functions, and involve an enormous number of MPI calls.

The test machine used for the ParMETIS case study was an HP Pavilion laptop running Ubuntu 8.10, with 4GB RAM and an Intel Core2Duo T-9300 CPU running at 2.5 GHz. The following are results from our GEM verifications:

- 2-procs: 1:58 min, 13,432 transitions, 1 interleaving
- 5-procs: 3:56 min, 71,633 transitions, 1 interleaving
- 10-procs: 9:36 min, 221,057 transitions, 1 interleaving

In all cases we found that over 95% of the total verification time was spent by ISP and the remainder was spent by GEM loading its data structures and displaying results. Through this case study we were better able to determine an upper bound on the scalability of our tool. Using ISP, GEM was able to scale to tens of processes.

4. CONCLUSIONS AND FUTURE PLANS

The limitations of GEM and ISP have necessitated work on a decentralized scheduling algorithm and in recent work, we have developed a scalable analysis tool called Distributed Analyzer of MPI Programs (DAMPI [6]) that conducts analysis similar to ISP, but with many scalability oriented heuristics that permit analysis to scale up to thousands of processes while running on a cluster. As part of our ongoing work we are in the process of seamlessly integrating DAMPI within GEM ([7]) to provide a framework for initial verification using ISP followed by subsequent verification of scaled up models using DAMPI. A hybrid scheme is also under consideration where known omissions of DAMPI (due to its scalability heuristics) are later analyzed through subsequent runs of ISP.

5. REFERENCES