Thesis Proposal: Verification of Habanero Java Programs

by

Radha Nakade

A thesis proposal, submitted to the faculty of Brigham Young University in partial fulfillment of the requirements for the degree of Master of Science.

Department of Computer Science

Brigham Young University

October 6, 2014

Abstract

Keep it around 150 words please.

1 Introduction

What are you going to do [**?**]?

2 Thesis Statement

Prove the basic properties (deadlock freedom, data race freedom, serialization and determinism) for task-parallel programming languages such as Habanero-Java, Cilk, X10, OpenMP 3.0, Chapel.

3 Related Work

Habanero java [1] is a task parallel programming language developed at the Rice University. It was developed as an extension to X10 with particular emphasis on safety properties of parallel constructs. The HJ compiler generates standard java class files that can run on any JVM. The HJ runtime is responsible for orchestrating the creation, execution and termination of HJ tasks.

HJ programs provide various safety guarantees if the parallel programming constructs are used correctly. Verifying HJ programs using tools such as Java Path Finder (JPF) can be time and memory consuming because of the numerous JPF state expansions. Hence, an HJ verification runtime (VR) [2] was developed at Brigham Young University to use JPF for verifying HJ programs. This runtime provides a lightweight alternative to verifying HJ programs using JPF.

Semantics of parallel programming languages:

Emmi and Boujjani introduced an interleaving free model of isolated hierarchical parallel computations for expressing general parallel programming languages [3]. They formalized a system for measuring the complexity of deciding state reachability for finite-data recursive programs.

Another way of creating formal models of programming languages is through Redex [4]. Redex is an executable domain-specific language for mechanizing semantic models developed by PLT. These models can be used to state theorems about the models and prove them. Redex is used by semantics engineers to formulate the syntax and semantics of the model, create test suites, run randomized testing and use graphical tools for visualizing examples etc.

Formalism of properties of Parallel Programming languages:

Scott and Lu have proposed various history-based definitions of determinism in [5]. They have discussed the comparative advantages of these defined properties. They have also discussed the containment relationships for these properties.

Dennis, Gao and Sarkar presented precise definitions of the two related properties of program schemata determinacy and repeatability in [6]. A key advantage of providing definitions for schemata rather than concrete programs is that it simplifies the task for programmers and tools to check these properties. The definitions of these properties are provided for schemata arising from data flow programs and task-parallel programs, thereby also establishing new relationships between the two models.

Race conditions occur in shared-memory parallel programs when accesses to shared-memory are not synchronized. A formal model to characterize different types of race conditions occurring in shared-memory parallel programs was developed by Netzer and Miller [7]. The race conditions are divided broadly into two categories - General races that cause deterministic programs to fail in execution and data races that appear in non-deterministic programs.

Banerjee et al. developed a rigorous mathematical framework that can be used to study the tradeoff between the amount of access history kept and the kinds of data races that can be detected [8]. Using this framework, they developed four algorithms for data race detection.

Bocchino et al. developed a region-based type and effect system for expressing important patterns of deterministic parallelism in imperative, object-oriented programs [9]. This system simplifies parallel programming by guaranteeing deterministic semantics with modular, compile time type checking.

Kahlon and Wang proposed a concept of Universal Causality Graphs (UCG) in [10]. UCGs encode the set of all feasible interleavings that a given correctness property may violate. UCGs provide a unified happens-before model by capturing causality constraints imposed by the property at hand as well as scheduling constraints imposed by synchronization primitives as causality constraints.

Checking Determinism:

In a parallel program, the threads of the parallel program can be interleaved non-deterministically during execution. Different thread interleavings result in different outputs for the same program input. Some of the results produced by such interleavings can be correct while others are wrong. Parallel programs should always produce the correct result irrespective of the thread interleavings that occur during program execution.

Burnim and Sen created an assertion framework that can be used to specify pairs of program state that can arise due to non-deterministic thread inter-leavings. Such pairs of program state result in a deterministic result in spite of the different parallel schedules. They created a java library that cab be used to specify these assertions. They also created an algorithm called Determin that can dynamically infer likely deterministic specification when provided with a set of inputs and schedules

Insta-check is another technique for checking external determinism during testing of parallel programs. It checks whether different runs of a parallel program with same input produce different outputs. This is done by computing a 64-bit hash of the memory state during program run. If two program runs with same input produce different hashes, then insta-check reports that the program is non-deterministic.

Vechev et al. developed a static analysis technique for automatic verification of determinism in parallel programs. The analysis is done in two phases. First phase identifies parts of the parallel program that run in parallel. Each part is sequentially analyzed by assuming that all memory locations accessed by the task are independent from locations accessed by other tasks that are running in parallel. In the second phase, the analysis checks whether this independence assumption holds i.e. all memory accesses are independent.

Data Race Detection:

Data races occur in parallel programs when two or more threads access a memory location and at least one of the accesses is a write. It is very difficult to detect data races in concurrent programs. A number of researchers have worked on data race detection.

Savage et al. developed a tool called Eraser to dynamically detect data races in multi-threaded programs. Eraser uses binary rewriting techniques to monitor every shared-memory reference and verify that consistent locking behavior is observed.

A method to perform static data race detection in concurrent C programs was developed by Kahlon et al. This method involved creating a precise context-sensitive concurrent control flow graph. Using this graph, identify the shared variables and lock pointers, compute on initial database of race warnings and then prune away the spurious messages using may-happen-in-parallel (MHP) analysis.

Flanagan and Freund developed a precise data race detection tool called FastTrack. It uses an adaptive lightweight representation for the happens-before relation that reduces both time and space overheads.

4 Project Description

Add your project description here.

5 Validation

Add stuff.

6 Thesis Schedule

The following schedule is an outline of submission dates for my Master's Thesis:
Submission to Advisor: February 8, 2014
Submission to Committee Members: February 22, 2014
Master's Thesis Defense: April 4, 2014

References

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis proposal submitted by

Eric S. Noonan

This thesis proposal has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Eric Mercer, Chair	Date
Jay McCarthy	Date
Mark Clement	Date
Quinn Snell, Graduate Coordinator	Date