Modeling and Verifying OpenMP Target Offloading Using CIVL

Akshar Patel
University of Illinois Chicago
Chicago, Illinois
apate776@uic.edu

Jan Hückelheim
Argonne National Laboratory
Lemont, Illinois
jhueckelheim@anl.gov

Zhiling Lan
University of Illinois Chicago
Chicago, Illinois Argonne National Laboratory
Lemont, Illinois
zlan@uic.edu

1 Introduction

Verification of concurrent programs is a long-existing and difficult problem due to the high levels of non-determinism that can be brought about with parallelism. Much work has been done to formally represent concurrent behaviors manually, but more recent attempts involve automating verification using model checking, static analysis, and symbolic execution. Regardless, the problem of verifying concurrent programs is only growing more complex as heterogeneous computing is becoming more prevalent, leading to more levels of parallelism to verify. In addition, the rise of LLMs have created plenty of LLM-generated and translated code with little tools to ensure that the generated code is functionally correct and that translated code is functionally equivalent to its source. OpenMP target offloading is one of many concurrency dialects for heterogeneous computing that need verification support. Although previous work exists in verifying OpenMP, they do not support target offloading which introduces new forms of complexity such as host-device execution and data-mapping semantics. An example of such a tool is CIVL or Concurrent Intermediate Verification Language. The goal of this work is to extend CIVL's OpenMP support by providing a model for representing OpenMP target offloading in CIVL's intermediate representation, CIVL-C. The work makes the following contributions...

(1) We create a base formal model in CIVL-C that models OpenMP target offloading
(2) We use that formal model to simulate multiple simple bugs in OpenMP and show that CIVL catches these bugs

2 Methodology

The formal model is based on an OpenMP target offloading directive that computes $Ax + B = C$ where A and B are input vectors, x is an input scalar, and C is the output vector. The OpenMP directive used was `#pragma omp target parallel loop map(to: A[0:n], B[0:n], x) map(from: C[0:n])`.

To create the formal model, three main technical challenges exist...

(1) **Host-device Execution Semantics**: How do we model the host and target device's separate execution in CIVL?
(2) **Data Mapping/Motion Semantics**: How do we map and transfer data between the host and device in accordance to the directive?
(3) **Thread Worksharing on the Device**: How do we preserve OpenMP's work partitioning and distribution semantics on the device while keeping track of data races?

2.1 Host-device Execution

One can model a host and device by implementing them as concurrently running CIVL processes. The host process handles non-directive instructions and non-offloaded directives while messages are sent to and from the host and device to run target offloaded directives. These messages act as a mechanism for transferring necessary data and synchronizing the host and device.

2.2 Data Mapping/Motion

Data that is mapped between the host and device can be copied over from the host to the device before directive execution and/or copied back from the device to the host after directive execution. This is modeled using CIVL's memory allocation and memory copy functions. The memory allocations and copies that occur are determined by the directive's `map` clause.

2.3 Thread Worksharing

Within the device process, multiple CIVL processes are spawned using CIVL's "parallel for" primitive. Each of these processes represents a thread being created by the OpenMP directive. Each of these threads are then given a portion of the OpenMP loop's iteration space that it must execute. Alongside the execution are functions maintaining read and write sets to check possible data races after thread execution.

3 Results

Using the formal model, we were able to simulate multiple simple bugs that one might see in this directive and find that CIVL was able to catch the simulated bug. For each bug, we first altered the original OpenMP directive so that the specified bug is present. We then altered our formal model so that it represents the changes made to the original directive. We then ran CIVL on that model and show that CIVL found the bug we added.

- **Thread-level Data Race**: forcing all iterations to write to $C[0]$ rather than their normal locations, creating a data race
- **Device-level Out of Bounds Memory Access**: Adding an extra iteration past the vector size to the directive's for-loop, forcing the last iteration to access memory that was not allocated to it
- **Missing Data in Map Clause**: Removing the A vector from the map clause, causing it to be inaccessible on the device
- **Function Specification Violation**: Creating a function specification for $Ax + B$ for the directive while changing the operator in the calculation from an addition to a multiplication.

4 Future Work

Given the formal model we developed, we can extend it to support more clauses in target offloading such as `nowait` and `teams distribute`. From there we can implement a frontend that extends CIVL's current OpenMP frontend with one that support target offloading. This will allow us to verify real scientific code bases that use OpenMP target offloading.