Modeling and Verifying OpenMP Target Offloading Using CIVL

Akshar Patel[1], Jan Hückelheim[2], Zhiling Lan[1,2]

[1]University of Illinois Chicago [2]Argonne National Laboratory

Motivation and Goal

- LLM generated and translate code is becoming more prevalent in scientific codebases
- Heterogeneous computing is also growing in popularity
- OpenMP target offloading is one of many concurrency dialects that can handle heterogeneous computing
- Formal verification can help ensure code is free of common bugs and is working as intended
- **Problem**: OpenMP target offloading introduces new semantics that most verification tools do not support despite supporting base OpenMP
- **Goal**: Develop a base formal model for OpenMP target offloading using CIVL's intermediate representation CIVL-C and verify basic correctness errors

OpenMP Target Offloading

```
#pragma omp target parallel loop \
    map(to: A[0:n], B[0:n], x) map(from: C[0:n])
for (int i = 0; i < n+1; i++) {
    C[i] = A[i] * x + B[i];
}
```

Contributions

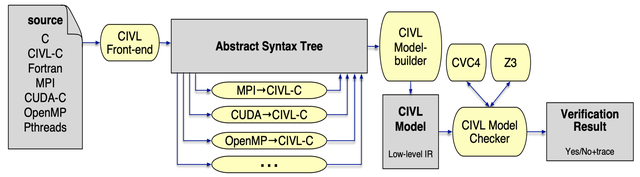
- We create a base formal model in CIVL-C that models OpenMP target offloading
- We use that formal model to simulate multiple simple bugs in OpenMP and show that CIVL catches these bugs

Technical Challenges

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?

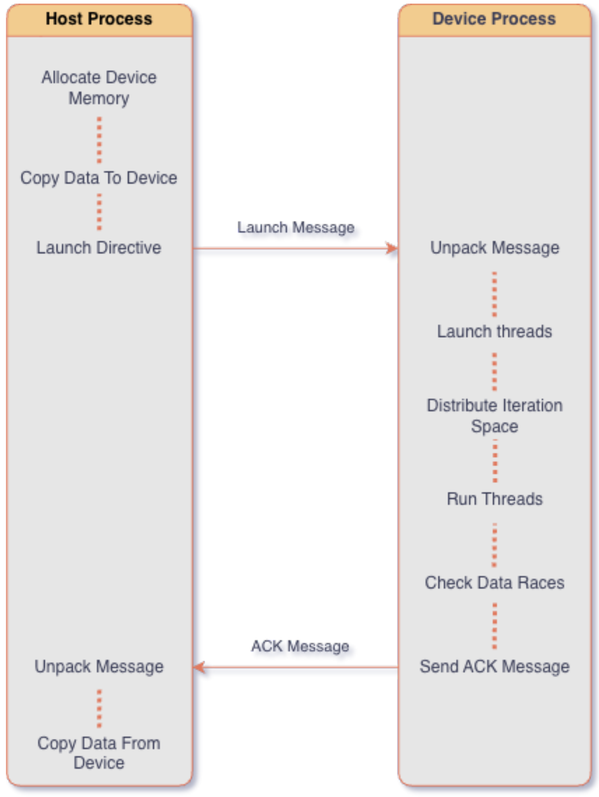
CIVL: Concurrent Intermediate Verification Language

- **CIVL** provides an IR, **CIVL-C**, alongside a verification backend that combines model checking, symbolic execution, and static analysis
- Support exists for base OpenMP, CUDA, Pthreads, and MPI
- Our approach involves translating an OpenMP target offloading example to CIVL-C



Methodology

- The host and device are represented by separate CIVL processes, sending messages back and forth to handle directive execution on the device
- The host performs memory copies to the device and back according to the what the map clause specifies
- The device launches the threads, distributes the iteration space among the threads, and executes the threads
- Read and write sets are used to check data races



Results

- Using the formal model, we were able to simulate multiple common bugs and show that CIVL finds them
- For each bug, the bug was first created in the original OpenMP directive
- Our formal model was then altered to represent the inserted bug
- The model was then given to CIVL to see if the bug would be detected
- Bug detection indicates that the formal model is representing the program correctly

Simulated Bugs in OpenMP Target Offloading

	OpenMP	CIVL-C	CIVL Output
Thread-level Data Race	C[i] => C[0] in for loop	C[i] => C[0] in for loop	Data-race detected violation
Missing Data in Map Clause	Remove A[] from map clause	Host does not copy A[] to the device	Undeclared identifier `A`
Out of Bounds Mem. Access	n => n+1 in for loop bounds (one extra loop)	n => n+1 in for loop bounds (one extra loop)	Illegal pointer dereference `A[i]`
Function Spec. Violation	Add function specification and change addition to multiplication in Ax + B	Insert same specification and change addition to multiplication in Ax + B	Assertion violation between expected and real result

Missing Data in Map Clause Example

OpenMP with bug:

```
#pragma omp target parallel loop \
    map(to: B[0:n], x) map(from: C[0:n])
for (int i = 0; i < n; i++) {
    C[i] = A[i] * x + B[i];
}
```

CIVL-C model before bug:

```
d_A = $hide((int*)$malloc(deviceScope, n * sizeof(int)));
d_B = $hide((int*)$malloc(deviceScope, n * sizeof(int)));
d_C = $hide((int*)$malloc(deviceScope, n * sizeof(int)));
d_x = $hide((int*)$malloc(deviceScope, sizeof(int)));

$host_memcpy(d_A, A, n * sizeof(int), MemcpyHostToDevice);
$host_memcpy(d_B, B, n * sizeof(int), MemcpyHostToDevice);
$host_memcpy(d_x, &x, sizeof(int), MemcpyHostToDevice);
```

CIVL-C model after bug:

```
d_B = $hide((int*)$malloc(deviceScope, n * sizeof(int)));
d_C = $hide((int*)$malloc(deviceScope, n * sizeof(int)));
d_x = $hide((int*)$malloc(deviceScope, sizeof(int)));

$host_memcpy(d_B, B, n * sizeof(int), MemcpyHostToDevice);
$host_memcpy(d_x, &x, sizeof(int), MemcpyHostToDevice);
```

CIVL Output:

```
CIVL v1.23 of 2026-05-26 --
https://civl.dev
Error: Undeclared identifier A at
A
AXPB_NM.cvl:116:17 | A
```

Future Work

- Extend the current formal model to support more clauses in OpenMP target offloading
  - Asynchronous offloading (*nowait)*
  - Explicit parallelism control (*teams distribute)*
  - Conditional offloading (*if*)
- Develop a full frontend on top of the current CIVL OpenMP frontend that supports target offloading in full
- Specify and verify real codebases that use OpenMP target offloading

Acknowledgments

This work was supported in part by the U.S. National Science Foundation under Grants Nos. OAC-2402901 and CCF-2515009 and in part by the U.S. Department of Energy under Contract No. DE-SC0024271. We also acknowledge support from the U.S. Department of Energy, Office of Science, under Grant No. DE-SC0023524 through the Chicagoland Computational Traineeship in High Energy Particle Physics Fellowship.