Arbalest: Dynamic Data Mapping Issue Detector for OpenMP Applications

Lechen Yu[1], Joachim Protze[2], Oscar Hernandez[3], Vivek Sarkar[1]

Georgia Tech[1], RWTH Aachen University[2], Oak Ridge National Laboratory[3]

EXASCALE COMPUTING PROJECT

OpenMP Target Offload & Data Mapping Issue

- Starting from version 4.0, OpenMP introduced the *target offload* feature into the specification to simplify the programming of accelerated applications
- Target offload is achieved by *device directives*
  - `target` construct - define compute kernel
  - `target data` construct - specify implicit data movement
  - `target update` construct - specify explicit data movement
  - `map` clause - specify data mapping
  - `nowait` clause - denote asynchronous compute kernel
- Incorrect usage of device directives may bring about *data mapping issues*, **namely memory synchronization errors on the host or device (e.g. stale data, uninitialized memory access)**

Examples From Open-source Benchmarks

DRACC_OMP_022_MxV_Missing_Data_yes[1] (Incorrect Mapping Type)

```
int a[N], b[N*N], c[N];

#pragma omp target
  map(to:a[0:N])
  map(alloc:b[0:N*N]) // should be "to"
  map(tofrom:c[0:N])
{
  #pragma omp teams distribute
  #pragma omp parallel for
    for(int i=0; i<N; i++)
      for(int j=0; j<N; j++)
        c[i]+=b[j+i*N]*a[j];
}

printf(c);
```

SPEC_ACCEL_503.postencil[2] (Missing Data Synchronization)

```
float h_A0[0:N], h_Anext[0:N];
#pragma omp target data
  map(to: h_A0[0:N])
  map(tofrom: h_Anext[0:N])
{
  for(int t=0;t<iteration;t++){
    // predefined compute kernel
    cpu_stencil(c0, c1, h_A0,
      h_Anext, nx, ny, nz);

    std::swap(h_A0, h_Anext);
    // miss #omp target update
    // for h_A0, h_Anext
  }
 }
// access h_A0 for odd iterations
print(h_Anext);
```

Related Work & Formalization

- There exists few tools capable of accurately and comprehensively detecting data mapping issues

Tool Name	Static / Dynamic	No False Positive	No False Negative
OMPSan[3]	Static	✗	✓
Archer[4]	Dynamic	✓	✗
ASan[5]	Dynamic	✓	✗
MSan[5]	Dynamic	✓	✗

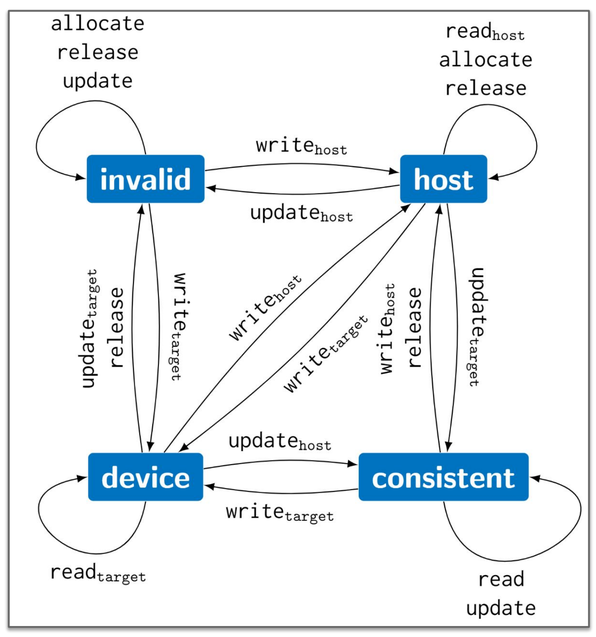
- We are the first to propose a formal definition of data mapping issue
  - The *shared-memory parallel version* (SPV) of an OpenMP application, which is acquired by removing all device directives, serves as the test oracle (the program having correct runtime behavior)
  - A data mapping issue occurs when an OpenMP program has a inconsistent dynamic data dependence with its SPV.

Precise Data Mapping Issue Detection

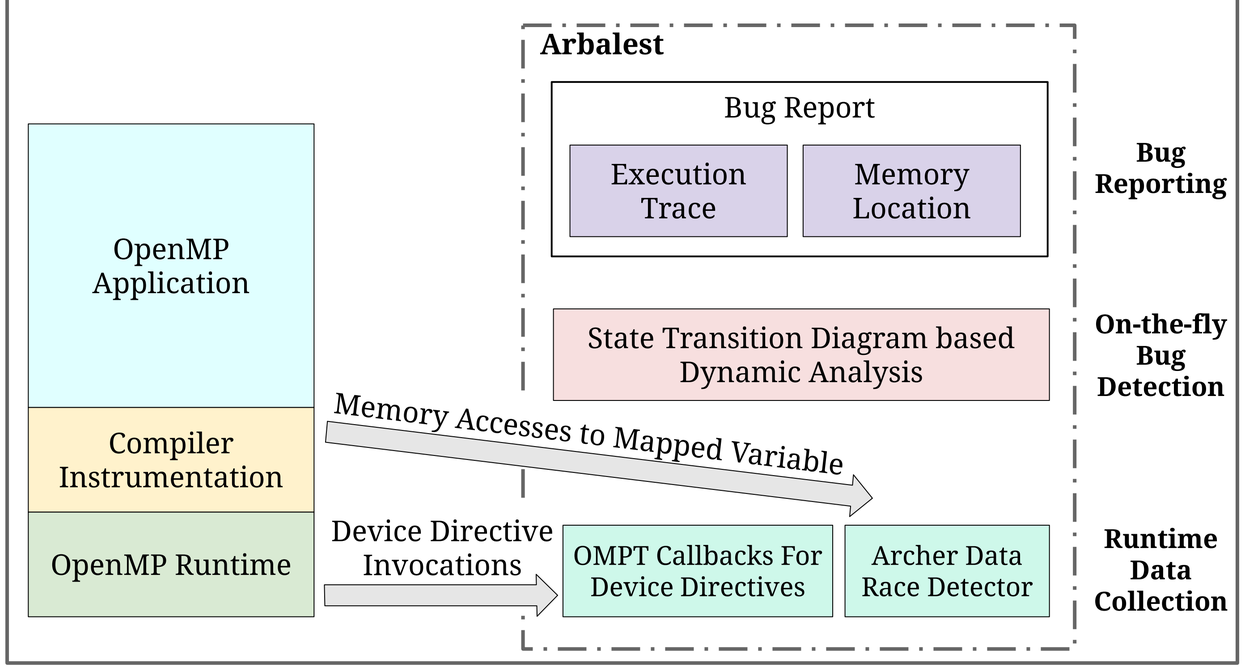
- Based on the formal definition, we further derive an equivalent correctness condition which is easy to examine at runtime
- For an OpenMP application without asynchronous kernels (without `nowait` clause), data-mapping-issue-free ⇔ each read observes the last write to the same variable (they may not operate on the same replica)
- For a general OpenMP application, data-mapping-issue-free ⇔ the OpenMP application does not incur any data mapping issue or data race when executing all asynchronous compute kernel synchronously
- Two main contributions of our data mapping issue detection method
  - general - compared to OMPSan, our method releases the constraints of serial elision
  - efficient - requires O(1) space per memory location and takes O(1) time for each memory access

Arbalest - Dynamic Data Mapping Issue Detector

- We have developed Arbalest, a dynamic data mapping issue detector
- Arbalest is an extension to Archer data race detector, reusing the infrastructure of Archer (e.g., shadow memory, OMPT callbacks)
- The data mapping issue detection is implemented as a state transition diagram
- Any access / update to the state transition diagram is lock-free



System Overview of Arbalest



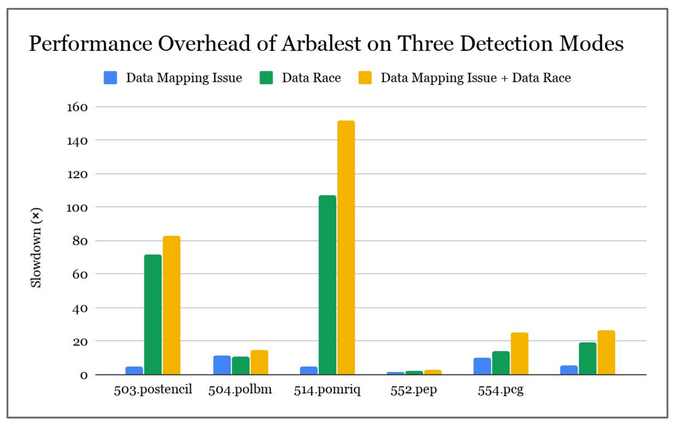
Evaluation - Correctness

- We conducted experiments on DRACC benchmarks to compare the effectiveness of Arbalest with other tools
- In total 13 benchmarks have data mapping issues
- The result indicates that Arbalest outperforms Archer, ASan, MSan on effectiveness

Benchmark	Effectiveness			
	Arbalest	Archer	TSan	MSan
DRACC_OMP_022	✓	✗	✗	✗
DRACC_OMP_024	✓	✗	✗	✗
DRACC_OMP_026	✓	✗	✗	✗
DRACC_OMP_027	✓	✗	✗	✗
DRACC_OMP_028	✓	✗	✗	✗
DRACC_OMP_030	✓	✓	✗	✗
DRACC_OMP_031	✓	✓	✗	✗
DRACC_OMP_032	✓	✗	✗	✗
DRACC_OMP_033	✗	✗	✗	✗
DRACC_OMP_034	✓	✗	✗	✗
DRACC_OMP_049	✓	✗	✗	✗
DRACC_OMP_050	✓	✗	✗	✗
DRACC_OMP_051	✓	✗	✗	✗

Evaluation - Performance

- We conducted experiments on SPEC-ACCEL benchmarks to measure the performance
- We executed Arbalest on three detection modes, comparing the overhead of data mapping issue detection and data race detection
- On average, Arbalest incurs 5.41×, 19.04×, and 26.75× slowdown to the native execution



[1]. DataRaceOnAccelerator (DRACC) Benchmark Suite. https://github.com/RWTH-HPC/DRACC
[2]. SPEC ACCEL® benchmark. https://www.spec.org/accel
[3]. OMPSan. http://parallel.auckland.ac.nz/iwomp2019/slides_OMPSan.pdf
[4]. Archer. https://github.com/PRUNERS/archer
[5]. Google Saniziters. https://github.com/google/sanitizers