Data Consistency Debugging for OpenMP Target Offload

Lechen Yu, Vivek Sarkar
Georgia Institute of Technology
{lechen.yu,vsarkar}@gatech.edu

Acknowledgments

- This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, in particular its subproject on Scaling OpenMP with LLVM for Exascale performance and portability (SOLLVE).

Background - OpenMP

Georgia Tech

- A popular parallel programming model for intra-node parallelism
- Supports multiple parallel paradigms
  - SPMD
  - Task parallelism
  - Heterogeneous parallelism
- For heterogeneous parallelism, OpenMP introduced device directives
  - Target constructs - compute kernel
  - Map clauses - data movement
- Observation: Optimal, or even correct, usage of OpenMP data mapping constructs can be non-trivial and error-prone

OpenMP Target Offloading

- Target - an abstraction of accelerator
  - Independent processing units
  - Separate memory space (if unified memory is not used)

- Target region - code region to be executed on a target
  - Declared by a target construct
  - Execution can be synchronous/asynchronous (nowait clauses)

- Data mapping - data movement to/from target device
  - Declared by map clauses

Example with target and map clauses

```c
int a = b = c = d = 0;

// kernel on the target
#pragma omp target   \
     map(alloc:a)    \
     map(to:b)       \
     map(from:c)     \
     map(tofrom:d)
{
     a = 1;
     b = 1;
     c = 1;
     d = 1;
}
```

Map-type	When to Synchronize Data	Semantics
alloc	Never	Allocate an uninitialized storage on the target
to	Enter the target region	Update the variable from host to target
from	Exit the target region	Update the variable from target to host
tofrom	Both	A combination of ‘to’ and ‘from’

Introduction - Consistency

- Result from incorrect usage of target constructs and map clauses
  - We refer to these errors as “data consistency” or “data mapping” issues

- Make a single variable have inconsistent values between the host and accelerator
  - Manually detecting and debugging data mapping issues can be challenging

- Data inconsistencies can lead to multiple kinds of memory issues
  - Use of Uninitialized Memory (UUM)
  - Use of Stale Data (USD)
  - Buffer Overflow (BO)
  - Data Race

Examples of Memory Issues that can result in Data Inconsistencies

- Example

```
1 int a[2] = {0, 0};
2 // a's map-type should be "tofrom"
3 #pragma omp target map(alloc:a[0:1])
4 {
5   a[0] = a[0] + 1;
6 }
7 print(a[0])
```

Use of uninitialized memory

Use of stale data

- line 5: when reading the copy on the device, it does not return the value of the original variable.
- line 7: when reading the original variable, it does not retrieve the updated value from the device.

Examples of Memory Issues that can result in Data Inconsistencies

- Buffer Overflow (BO) resulting from data mapping issues

```c
int a[2] = {0, 0};
// the array section should be
// a[0:2]
#pragma omp target     \
    map(tofrom:a[0:1])
{
   a[1] = a[1] + 1;
}
print(a[1])
```

Buffer Overflow

- Data Race resulting from data mapping issues

```c
int a[2] = {0, 0};
// the target region executes
// asynchronously
#pragma omp target nowait \
    map(tofrom:a[0:2])
{
   a[1] = a[1] + 1;
}
print(a[1])
```

Data Race

Outline for the rest of the talk

- Static analysis of memory consistency errors
  - OMPSan: Static Verification of OpenMP's Data Mapping Constructs *[IWOMP 2019]*

- Dynamic analysis of memory consistency errors
  - ARBALEST: Dynamic Detection of Data Mapping Issues in Heterogeneous OpenMP Applications *[IPDPS 2021]*

OmpSan - Static Data Inconsistency Detector

Prithayan Barua, J. Shirako, Whitney Tsang, Jeeva Paudel, Wang Chen, Vivek Sarkar. "OMPSan: Static Verification of OpenMP's Data Mapping Constructs" IWOMP 2019 (Recipient of Best Paper Award)

Paper link: https://link.springer.com/chapter/10.1007/978-3-030-28596-8_1

OmpSan - How to Formally Define Data Inconsistency

Georgia Tech

- Assumption: OpenMP application is expected to yield the same result if all OpenMP directives are ignored (serial-elision property)
  - Desirable property for most OpenMP programs
- Definition:
  - A data inconsistency occurs when there exists a different def-use relation between the OpenMP program and its serial-elision version (the same program with all OpenMP constructs ignored)

OmpSan: compare use-def relations in OpenMP vs. sequential code

```
int a = 0;

#pragma omp target \
        map(alloc:a)
{
  a = 1;
}


print(a)
```

```
host_array_write(A)
        ↓
copy_host_to_device(A)
        ↓
device_array_write(A)
        ↓

        ↓
host_array_read(A)
```

OpenMP def-use

$$A_1^{host} = d\phi(A_0^{host})$$

$$A_1^{device} = mcpy\phi(A_1^{host})$$

$$A_2^{device} = d\phi(A_1^{device})$$

$$A_2^{host} = u\phi(A_1^{host})$$

Sequential def-use

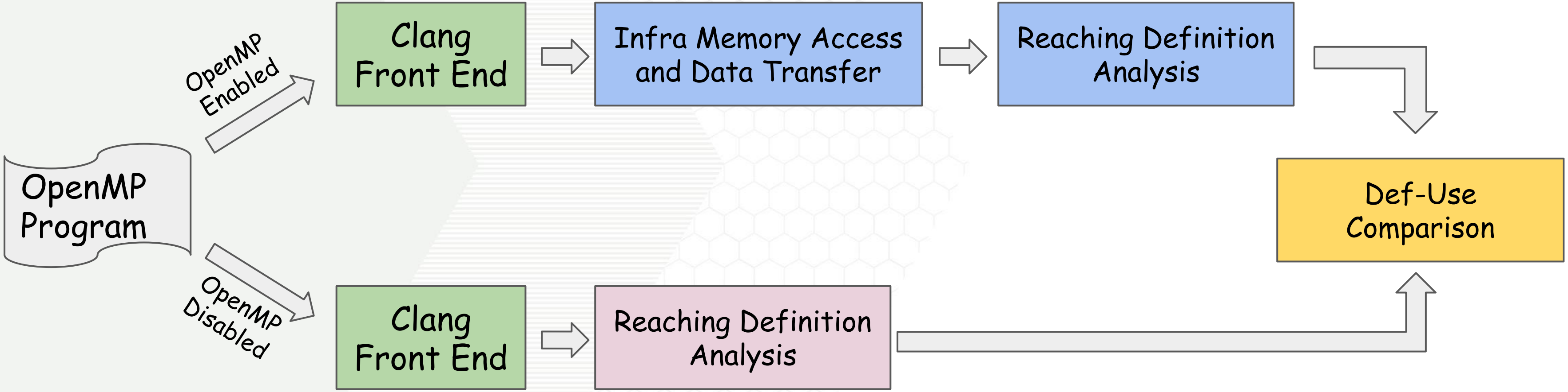
$$A_1 = d\phi(A_0)$$

$$A_2 = d\phi(A_1)$$

$$A_3 = d\phi(A_2)$$

$$A_4 = u\phi(A_3)$$

OmpSan - Implementation



OmpSan - Evaluation

- Benchmarks
  - DataRaceOnAccelerator (DRACC)
    - Micro-benchmark suite for OpenMP
    - Designed to evaluate correctness tools' capabilities on detecting memory issues
    - 16 micro-benchmarks have data inconsistency
  - SPEC ACCEL 1.3
    - Performance benchmark suite for OpenMP target offloading
  - Ported NAS parallel benchmark
    - Performance benchmark suite originally using OpenMP SPMD constructs
    - Ported by the research team at University of Delaware

DRACC: https://github.com/RWTH-HPC/DRACC

OmpSan - Evaluation

Benchmark	Result
DRACC	Detected all 16 data inconsistency
SPEC-ACCEL	No false positive reported
NAS	No false positive reported

Outline for the rest of the talk

Georgia Tech

- Static analysis of memory consistency errors
    - OMPSan: Static Verification of OpenMP's Data Mapping Constructs *[IWOMP 2019]*

- Dynamic analysis of memory consistency errors
    - ARBALEST: Dynamic Detection of Data Mapping Issues in Heterogeneous OpenMP Applications *[IPDPS 2021]*

ARBALEST - Dynamic Data Inconsistency Detector

Lechen Yu, Joachim Protze, Oscar Hernandez, and Vivek Sarkar. "ARBALEST: Dynamic detection of data mapping issues in heterogeneous openmp applications"

Paper Link: https://ieeexplore.ieee.org/document/9460498

ARBALEST - Introduction

Georgia Tech

- Static analysis tools may be incapable of tackling all categories of data inconsistency
  - Some of them require the precise runtime information
    - Array bound
    - Reference count

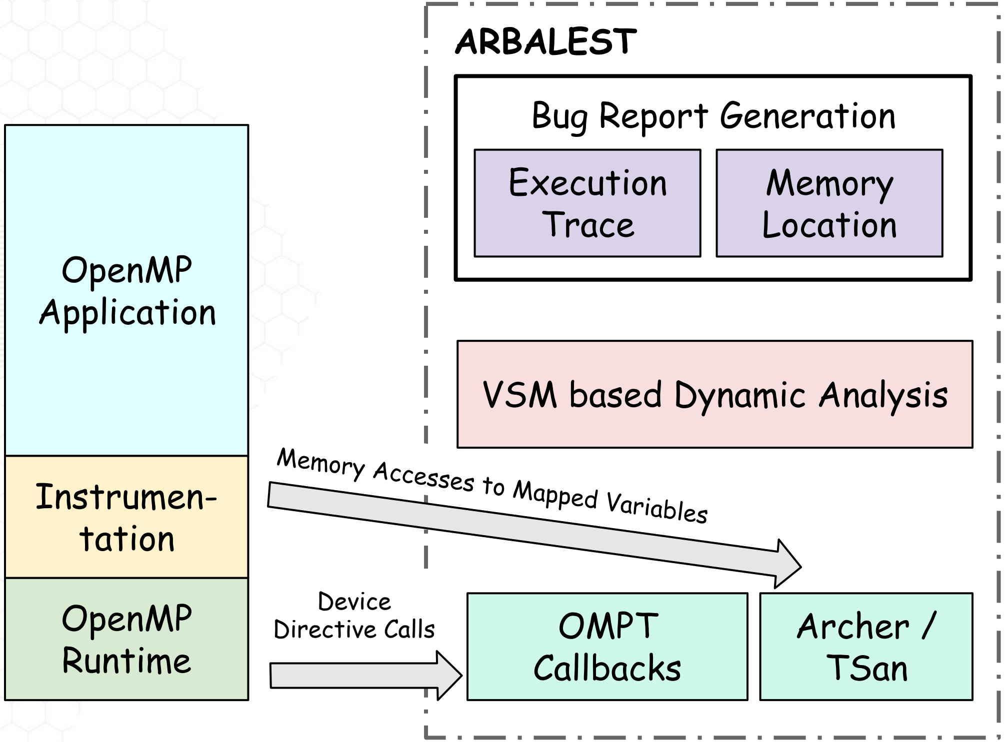
- To the best of our knowledge, existing dynamic analysis tools can only detect a subset of data inconsistency
  - No dynamic analysis tool is designed for data inconsistency in OpenMP programs

ARBALEST - Introduction

- A dynamic data inconsistency detector for OpenMP programs
- Built on top of Archer, the state-of-the-art OpenMP data race detector
- Extend Archer's infrastructure to tackle all four categories of data inconsistency

ARBALEST

- Extends Archer Race Detector for OpenMP programs executing on multicore CPU
  - Use host as a virtual "device" to simulate target offloading
- Launches all target regions on the host
- Use OpenMP Tool interface (OMPT) to intercept OpenMP construct calls
- IPDPS 21's implementation uses LLVM 9.0

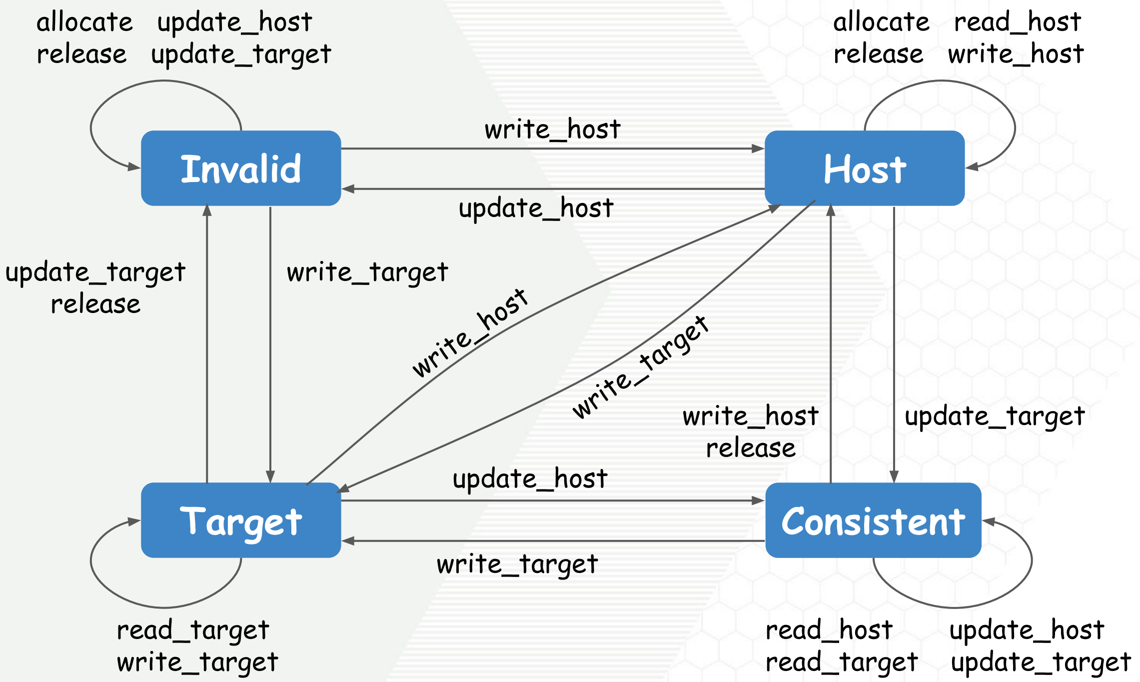


1. Archer: https://github.com/llvm/llvm-project/tree/main/openmp/tools/archer

VSM (Variable State Machine)

- A state machine to track the validity of each variable
- A read/write/data movement triggers the state transition
- At any state, a read operation having no corresponding state transition indicates a data inconsistency

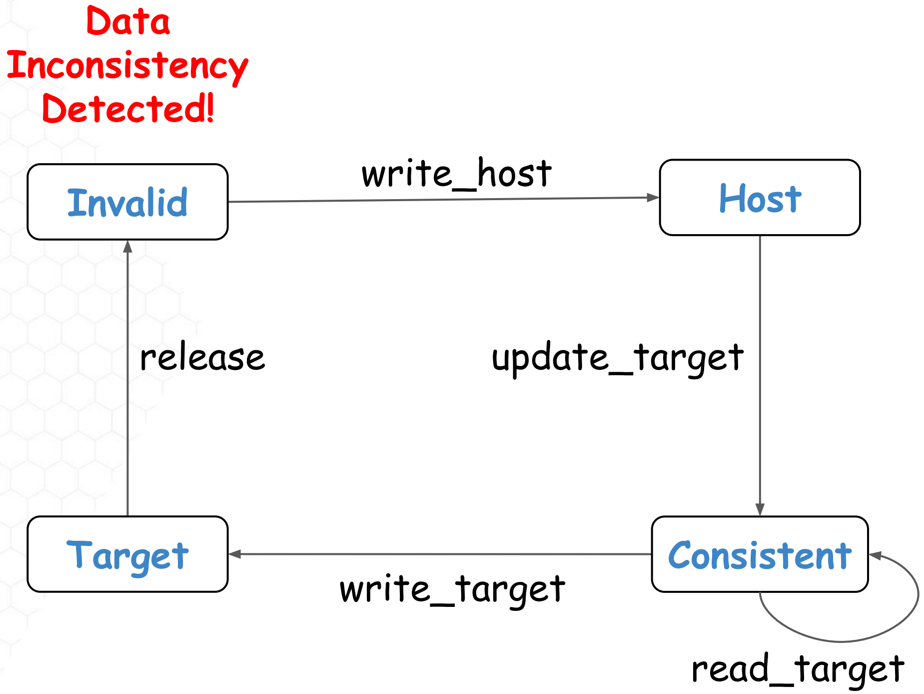
VSM



State	Device with Valid Value	Illegal Operations
Invalid	None	read_host read_target
Host	Host	read_target
Target	Target	read_host
Consistent	Both	None

VSM Examples

```
int a[2] = {0, 0};
// a's map-type should be "tofrom"
#pragma omp target map(to:a[0:2])
{
    a[0] = a[0] + 1;
}
print(a[0])
```



Add VSM into Archer

- Reserve four bits of Archer's shadow state for VSM
- First two bits represent the state in VSM
- Remaining two bits are used to distinguish UUM from USD

Field	Size
IsOVValid	1 bit
IsCVValid	1 bit
IsOVInitialized	1 bit
IsCVInitialized	1 bit
TID (Thread Id)	12 bits
Scalar Clock	42 bits
IsWrite	1 bit
Access Size (1, 2, 4 or 8)	2 bits
Address Offset (0..7)	3 bits

Sanitizer Shadow State: https://github.com/google/sanitizers/wiki/ThreadSanitizerAlgorithm#shadow-state

Other modifications to Archer

- Revise Archer's instrumentation pass to add a boundary check for array access
  - Capture all pointer arithmetic on array / pointer
  - At runtime, examine whether the result and the base pointer belong to the same mapped variable

- Annotate data movement as read/write to shared memory on the host
  - Host-to-target: read operations
  - Target-to-host: write operations
  - Construct shadow words for read/write and invoke Archer's race detection routine

Evaluation Setup

- Evaluated precision and performance using two sets of benchmarks
  - Compared with four dynamic analysis tools
    - AddressSanitizer
    - MemorySanitizer
    - Archer
    - Valgrind
- These tools can capture a subset of data inconsistency
  - Designed for memory issues
  - May report a data inconsistency when the bug further leads to memory issues they can detect

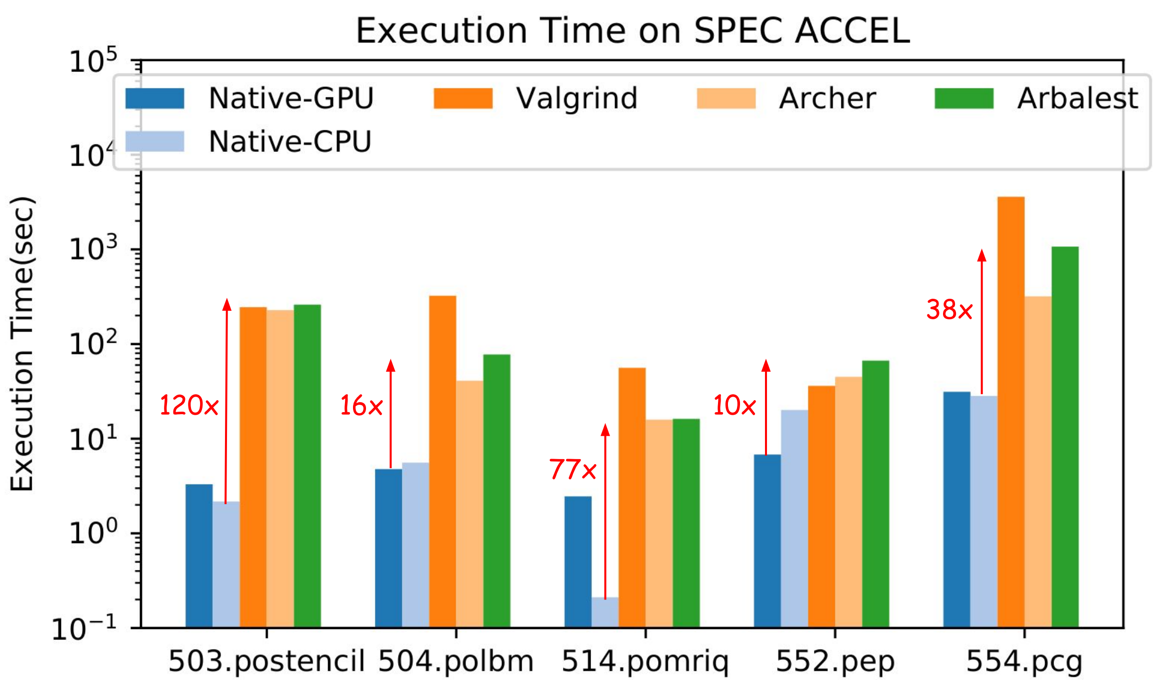
Evaluation Setup

- Precision evaluations are conducted on DRACC benchmark suite
  - 56 OpenMP micro-benchmarks designed for precision evaluation
  - 16 out of 56 micro-benchmarks have data inconsistency
- Performance evaluations are conducted on SPEC-ACCEL 1.2 benchmark suite
  - 'test' input was used for these evaluations
  - All tools execute OpenMP programs on the CPU when debugging data inconsistency/memory issues

Precision Evaluations

Benchmark ID	Effect	Effectiveness				
		Arbalest	Valgrind	Archer	ASan	MSan
22, 24, 49, 50, 51	UUM	✓	-	-	-	✓
23, 25, 28, 29, 30, 31	BO	✓	✓	-	✓	-
26, 27, 32, 33, 34	USD	✓	-	-	-	-
Overall		16/16	6/16	0/16	6/16	5/16

Performance Evaluations - Time Overhead



Why ARBALEST does more checks than Archer

- Archer

```
1  int a[2] = {0, 0};
2
3  #pragma omp target    \
4      map(to:a[0:2])
5  {
6     a[1] = a[1] + 1;
7  }
8
9  print(a[1])
10
11 a[1] = 0
```

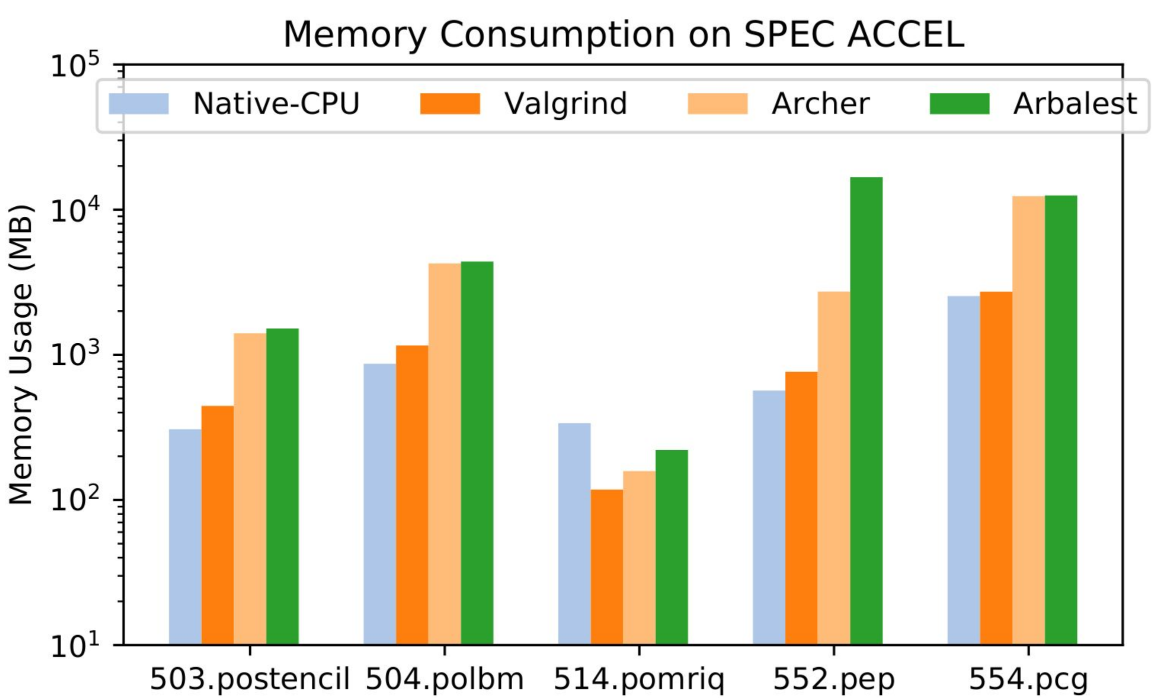
- read in line 9 will be skipped by Archer's instrument pass because the succeeding write in line 10

- ARBALEST

```
1  int a[2] = {0, 0};
2
3  #pragma omp target    \
4      map(to:a[0:2])
5  {
6     a[1] = a[1] + 1;
7  }
8  // check VSM
9  print(a[1])
10 // check race & update VSM
11 a[1] = 0
```

- ARBALEST instruments both line 9 and line 11, carrying out more checks

Performance Evaluations - Space Overhead



Work in Progress - Implement ARBALEST in LLVM 15

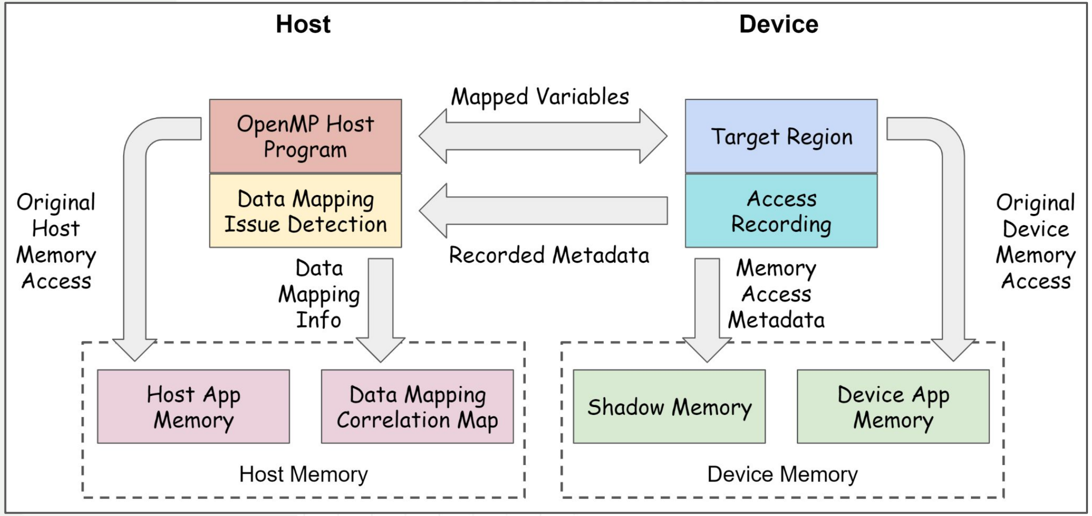
- We have built a new prototype of ARBALEST on top of LLVM 15
- Make ARBALEST compatible with those new features in LLVM Sanitizers
  - Archer/ThreadSanitizer in LLVM 15 uses SSE2 instruction set to accelerate data race detection
  - Uses a concise shadow memory layout, reducing shadow state's size from 64 bits to 32 bits
  - We have successfully embed VSM into the new format of Archer's shadow word
- Introduce a new OMPT event to better model the behavior of a map clause
  - e.g., target map(to: array) indicates three target-data-op events for array
  - allocation, association, and data transfer
  - using a single event to record all data ops related to a map clause

Get Access to These Tools

- The prototype of these two tools are hosted on Georgia Tech's GitHub Enterprise
- We are testing them with more benchmarks and real-world OpenMP applications
- Please email us if you want to get access

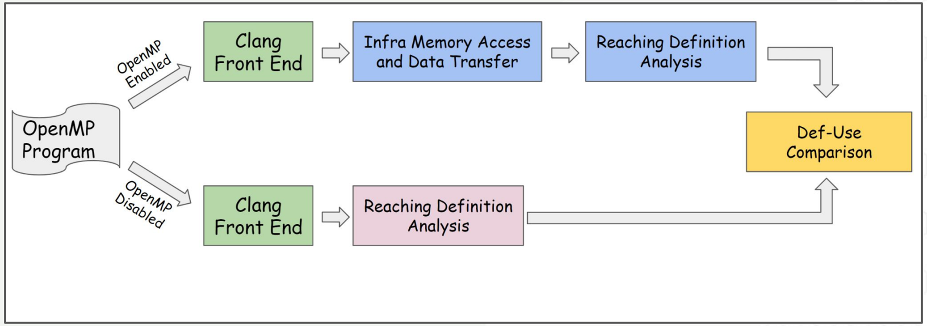
Work in Progress

- Explore the probability of detecting data inconsistency on the native device, e.g., GPU



Takeaway

- OmpSan - Static Data Inconsistency Detector



- ARBALEST - Dynamic Data Inconsistency Detector