A Study of Memory Anomalies in OpenMP Applications

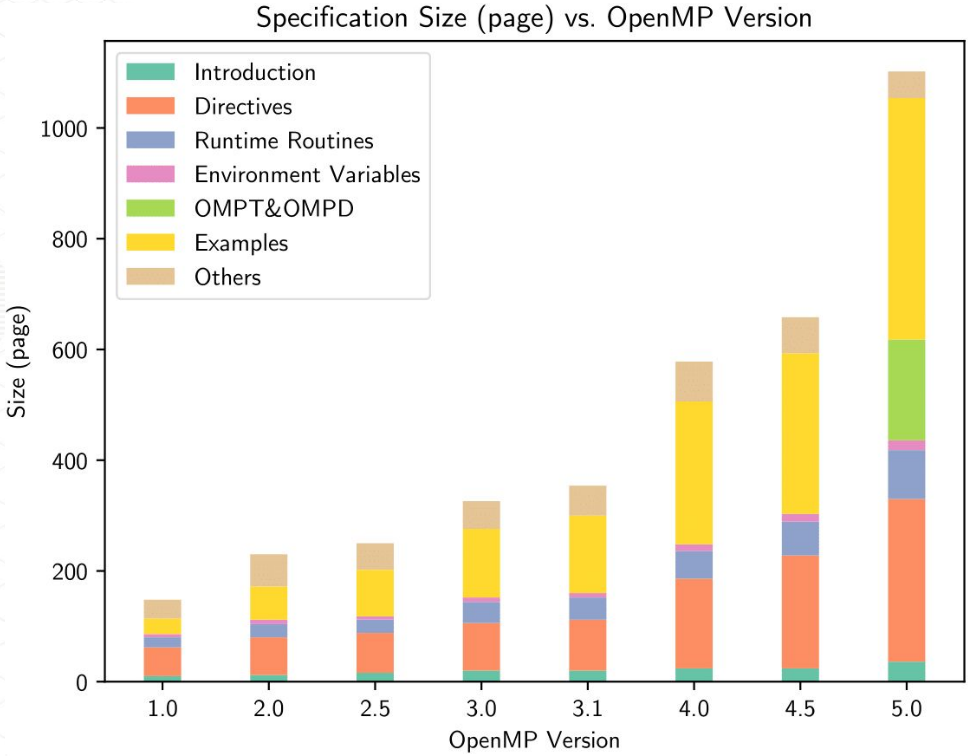
Lechen Yu, Joachim Protze, Oscar Hernandez, Vivek Sarkar

Introduction & Motivation

Trend of OpenMP Specification

- OpenMP has supported multiple parallel paradigms
    - SPMD
    - Task parallelism
    - Heterogeneous parallelism

- When introducing a new parallel paradigm, the size of the specification increases significantly.



Memory Anomalies

- Memory anomalies are common bugs in C/C++ applications[1]
  - Use of Uninitialized Memory (UUM)
  - Use of Stale Data (USD)
  - Use After Free (UAF)
  - Buffer Overflow (BO)

- Manually detecting memory anomalies is a cumbersome task
  - Memory anomalies may lead to numerous unexpected runtime behavior
    - Silent error
    - Undefined behavior
    - Program crash
  - The root cause may be far removed from the point where the bug becomes apparent

1. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43308.pdf

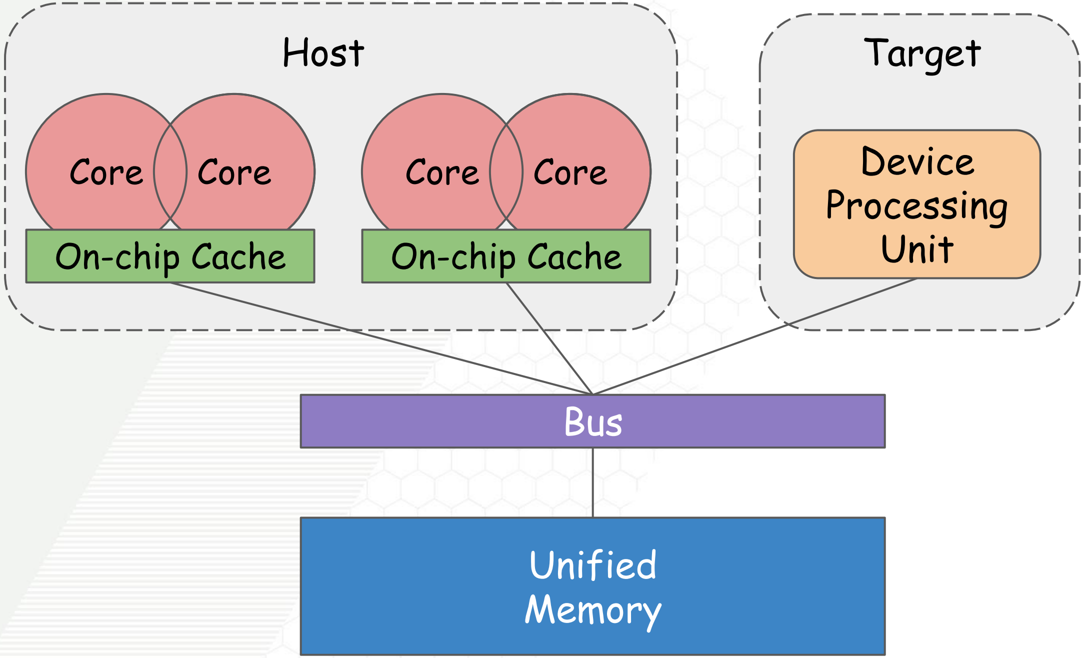
Our Work and Contribution

- To the best of our knowledge, there exists no prior work focusing on memory anomalies in OpenMP applications
  - Unlike data races, our focus on memory anomalies identifies bugs that can occur in sequential or parallel execution
- We conducted a study on memory anomalies resulting from incorrect usage of OpenMP constructs
  - Incorrect setting of data-sharing attribute
  - Incorrect setting of map-type
- We also carried out an evaluation on three state-of-the-art memory anomaly detectors
  - AddressSanitizer (ASan), MemorySanitizer (MSan), Valgrind

Background

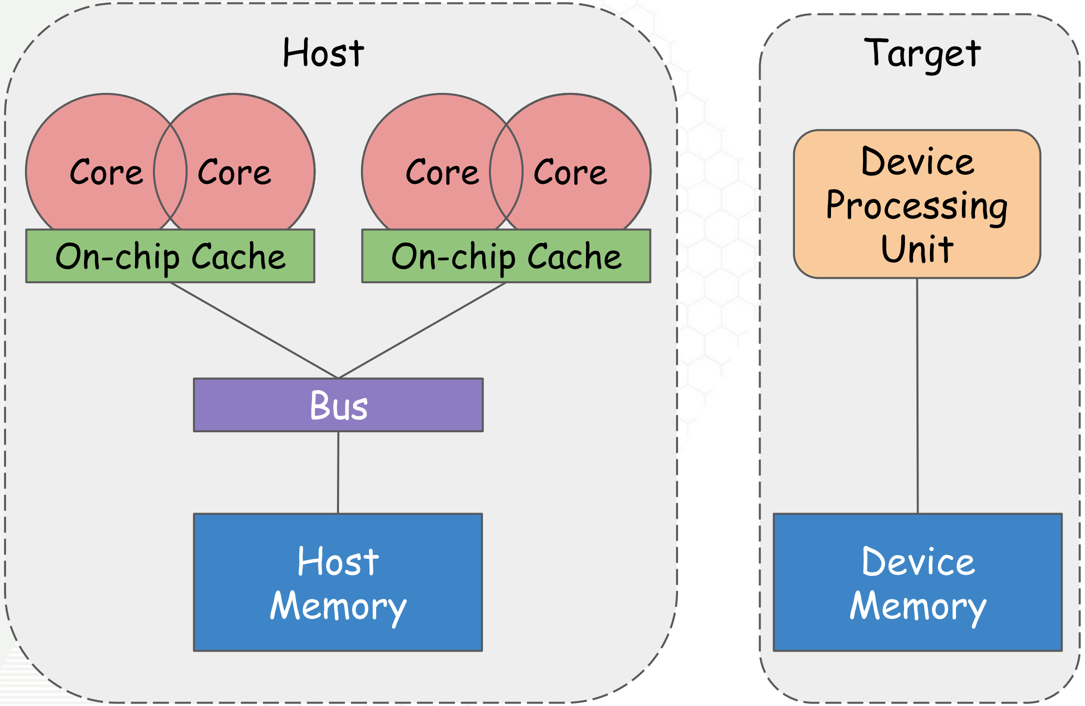
OpenMP's Execution and Memory Model

- Unified Memory



OpenMP's Execution and Memory Model

- Separate Memory



Data-sharing Attributes in OpenMP

Georgia Tech

- Data-sharing attributes impact physical locations used to store accessed variables in an OpenMP construct

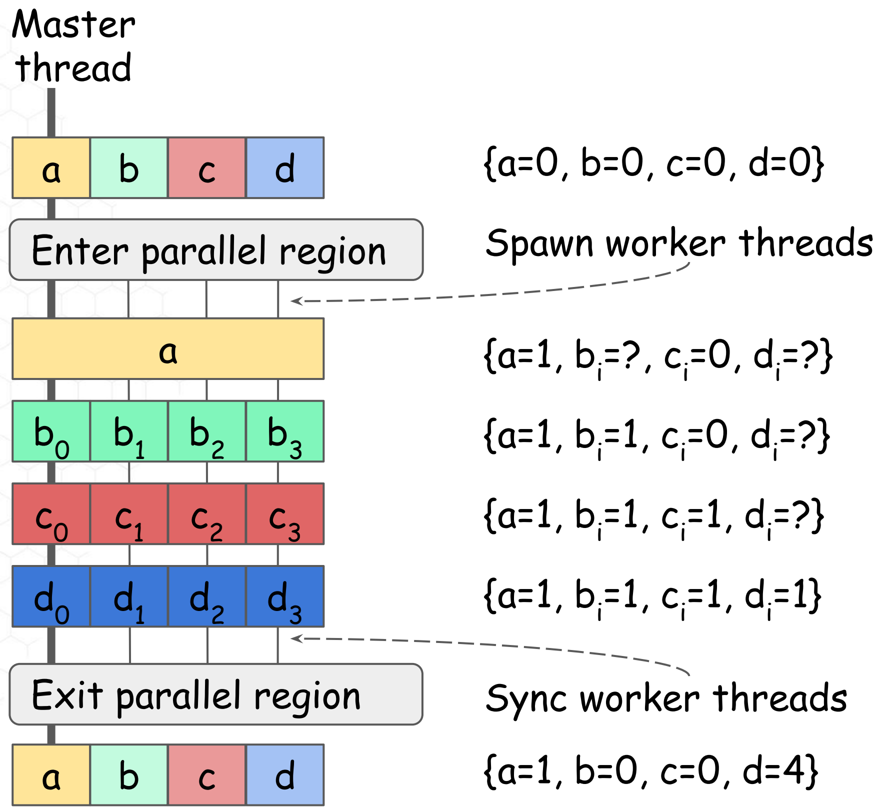
- Data-sharing attributes also affect the values of accessed variables

Data-sharing Attributes in OpenMP

Georgia Tech

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

#pragma omp parallel num_threads(4) \
    shared(a)                       \
    private(b)                      \
    firstprivate(c)                 \
    reduction(+:d)
{
    a = 1;
    b = 1;
    c = 1;
    d = 1;
}
```



Map-types in OpenMP

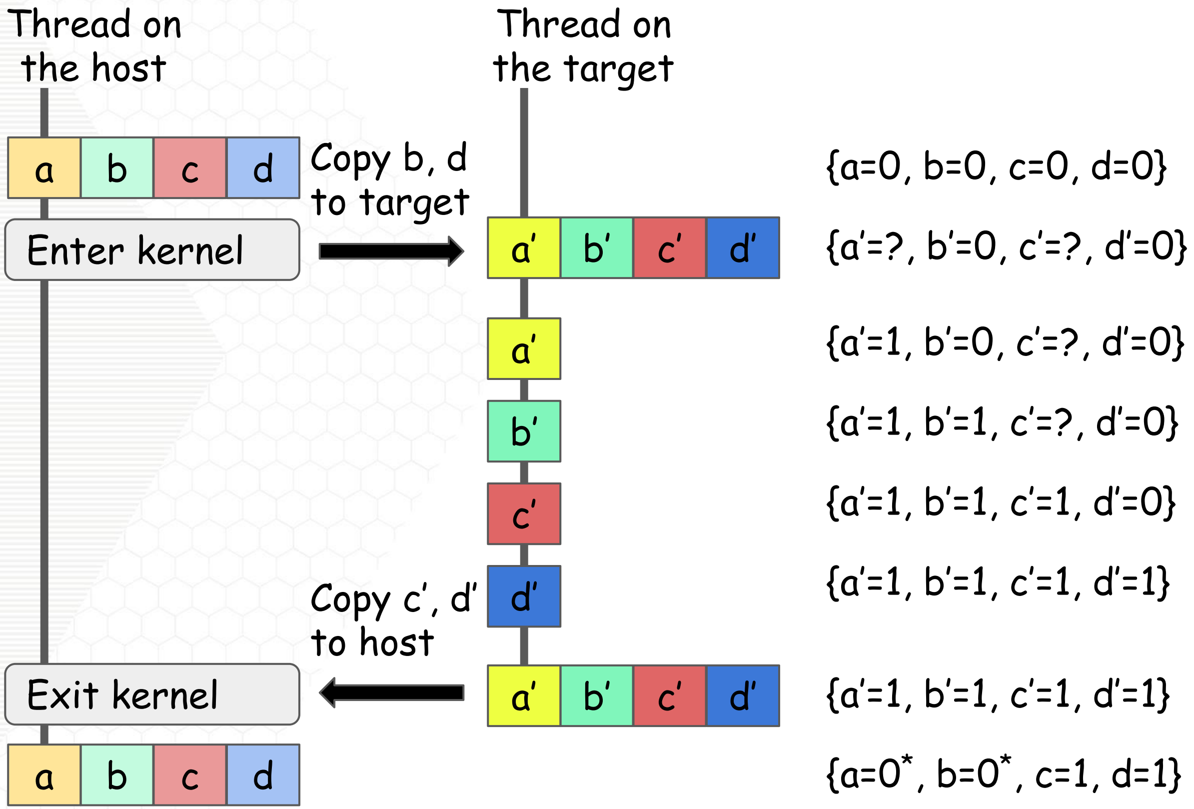
- Map-types declare data transfers between the host and target
- All map-types in OpenMP

Map-type	When to take effect	Semantics
to	Enter the target region	Copy the variable from host to target
alloc	Enter the target region	Allocate an uninitialized storage on the target
from	Exit the target region	Copy the variable from the target to host
delete/release	Exit the target region	Deallocate the storage on the target
tofrom	Both	A combination of 'to' and 'from'

Map-types in OpenMP

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

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



Memory Anomalies in OpenMP Applications

Questions to Answer

- Q1 (bug pattern): For memory anomalies in OpenMP, what are the common bug patterns and root causes?
- Q2 (bug fix): How to fix these memory anomalies in OpenMP?
- Q3 (tool effectiveness): What is the effectiveness of state-of-art memory anomaly detectors on OpenMP applications?

Use of Uninitialized Memory (UUM)

- UUM resulting from incorrect data-sharing attribute

```
int count = 0;

#pragma omp parallel for
    private(counter)
for (int i=0; i<N; i++){
    #pragma omp atomic update
    counter++;  UUM
}
```

- UUM resulting from read semantics of OpenMP clauses

```
// sum is uninitialized
int sum;
int SIZE = 512;
int a[SIZE];
memset(a, 1, SIZE * sizeof(int));                UUM

#pragma omp parallel for reduction(+:sum)
for(int i = 0; i < SIZE; i++) {
    sum+=a[i];
}
```

Use of Uninitialized Memory (UUM)

- UUM resulting from incorrect map-type

```
#define N 512
int a[N], b[N*N], c[N];
memset(a, 2, SIZE * sizeof(int));
memset(b, 2, SIZE * SIZE * sizeof(int));
memset(c, 0, SIZE * sizeof(int));
// b's map-type should be "to”
#pragma omp target                          \
    map(to:a[0:N]) map(alloc:b[0:N*N])      \
    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];    UUM
}
```

Use of Stale Data (USD)

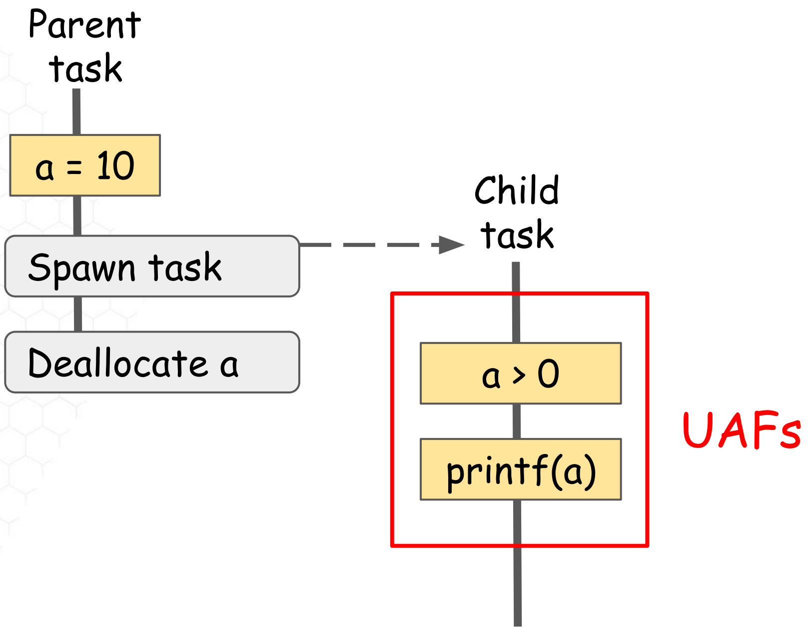
- USD resulting from incorrect map-type

```
int SIZE = 5000;
int a[SIZE], b[SIZE];
memset(a, 0, SIZE * sizeof(int));
memset(b, 1, SIZE * sizeof(int));

// a's map-type should be 'from' or `tofrom'
#pragma omp target \
    map(to: a[0:SIZE]) map(to: b[0:SIZE])
{
  #pragma omp teams distribute
  #pragma omp parallel for
  for (int i = 0; i < SIZE; i++)
    a[i] = b[i] + b[i];
}
printf(a[0]);  USD
```

Use After Free (UAF)

```
#pragma omp parallel
#pragma omp single
#pragma omp task
{
  int a = 10;
  #pragma omp task shared(a)
  {
    // 'a' may have been released
    if (a > 0)
      printf(a);
  }
  // FIX: add a 'taskwait' here
}
```



Evaluations of Memory Anomaly Detectors

Challenge for Memory Anomaly Detectors

Georgia Tech

- None of the memory anomaly detectors are designed for OpenMP applications
- Memory anomaly detectors need to correctly model the semantics of OpenMP constructs
- Existing tools (e.g., LLVM Sanitizer) may report false positives/false negatives when tackling OpenMP applications

```
// sum is uninitialized
int sum = 0;
int SIZE = 512;
int a[SIZE];
memset(a, 1, SIZE * sizeof(int));

#pragma omp parallel for \
    reduction(+:sum)
for(int i = 0; i < SIZE; i++) {
    sum+=a[i];
}
```

May report false positive

Evaluation Setup

- Evaluated Memory Anomaly Detectors
  - AddressSanitizer (ASan) in LLVM 10.0
  - MemorySanitizer (MSan) in LLVM 10.0
  - Valgrind memcheck in Valgrind 3.14.0
- Benchmarks
  - In total 22 benchmarks, each of which contains a memory anomaly
  - 15 map-type-related benchmarks are from DRACC[1]
  - 7 data-sharing-attribute-related benchmarks are constructed based on our experience[2]
- OS & Compiler
  - Evaluations are carried out on a compute node of CLAIX cluster running CentOS 7
  - All 22 micro-benchmarks are compiled with LLVM 10.0

1. https://github.com/RWTH-HPC/DRACC 2. https://github.com/FuriousBerserker/MemoryAnomalyBench

Results of 15 DRACC Benchmarks

Georgia Tech

- Valgrind outperforms ASan and MSan, but none of them can tackle all memory anomalies
- ASan only reports buffer overflows and MSan only reports UUMs
- None of them can tackle USDs

Benchmark	Error	Effectiveness		
		ASan	MSan	Valgrind
DRACC_OMP_022	UUM	✗	✓	✗
DRACC_OMP_023	BO	✓	✗	✓
DRACC_OMP_024	UUM	✗	✓	✗
DRACC_OMP_025	BO	✓	✗	✓
DRACC_OMP_026	USD	✗	✗	✗
DRACC_OMP_027	USD	✗	✗	✗
DRACC_OMP_028	BO	✓	✗	✓
DRACC_OMP_029	BO	✓	✗	✓
DRACC_OMP_030	BO	✓	✗	✓
DRACC_OMP_031	BO	✓	✗	✓
DRACC_OMP_032	USD	✗	✗	✗
DRACC_OMP_033	USD	✗	✗	✗
DRACC_OMP_049	UUM	✗	✓	✓
DRACC_OMP_050	UUM	✗	✓	✓
DRACC_OMP_051	UUM	✗	✓	✓
Overall		6/15	5/15	9/15

Results of the Other Seven Benchmarks

- Valgrind outperforms ASan and MSan, but none of them can tackle all memory anomalies
- ASan does not report any memory anomalies
- MSan misses two UUMs due to the underlying detection algorithm (UUM is reported when the variable is used by a code branch)

Benchmark	Error	Effectiveness		
		ASan	MSan	Valgrind
DSA_OMP_001	UUM	✗	✓	✓
DSA_OMP_002	UUM	✗	✓	✓
DSA_OMP_003	UUM	✗	✗	✓
DSA_OMP_004	UUM	✗	✓	✓
DSA_OMP_005	UUM	✗	✗	✓
DSA_OMP_006	UAF	✗	✓	✗
DSA_OMP_007	USD	✗	✗	✗
Overall		**0/7**	**4/7**	**5/7**

DSA_OMP_005

- UUM in DSA_OMP_005

```
1 int countervar = 0;
2 #pragma omp parallel for
3     private(countervar)
4 for (int i = 0; i < N; i++) {
5     #pragma omp atomic update
6     countervar++;
7 }
```

Undetected UUM

- Corresponding IR for line 5

```
%217 = ptrtoint i32* %countervar to i64
%218 = xor i64 %217, 87960930222080
%219 = inttoptr i64 %218 to i32*
%220 = add i64 %218, 17592186044416
%221 = and i64 %220, -4
%222 = inttoptr i64 %221 to i32*
store i32 0, i32* %219, align 4
%223 = atomicrmw add i32* %countervar, i32 1 release
```

MSan does not instrument this write!

Future Work

- Evaluate the performance of state-of-the-art memory anomaly detectors
- Compare the quality of bug reports from different memory anomaly detectors
- Develop a memory anomaly detector for OpenMP applications, which covers a larger set of memory anomalies compared to existing tools

Takeaways

- OpenMP applications may encounter memory anomalies if programers use incorrect data-sharing attributes or map-types
- OpenMP applications may encounter numerous types of memory anomalies, including UUM, USD, UAF, and BO
- Existing memory anomaly detectors can only detect a subset of memory anomalies in an OpenMP application

Backup Slides

Data-sharing Attributes in OpenMP

<table>
<tr><th colspan="2">Precondition</th><th>Rule Type</th><th>Data-Sharing Attribute</th></tr>
<tr><td colspan="2">Declared inside a construct</td><td>PRE</td><td>private</td></tr>
<tr><td colspan="2">Static class member, objects with dynamic storage duration</td><td>PRE</td><td>shared</td></tr>
<tr><td colspan="2">A loop iteration variable in a <code>for</code>, <code>parallel for</code>, <code>task loop</code>, or <code>distribute</code> construct</td><td>PRE</td><td>private</td></tr>
<tr><td colspan="2">A loop iteration variable in a <code>simd</code> or <code>loop</code> construct</td><td>PRE</td><td>last-private</td></tr>
<tr><td colspan="2">Listed in a <code>reduction</code> clause</td><td>EXP</td><td>private</td></tr>
<tr><td colspan="2">Listed in a data-sharing attribute clause (programmer-specified data-sharing attribute)</td><td>EXP</td><td>Determined by the clause</td></tr>
<tr><td colspan="2">An unmapped variable in a <code>target</code> construct</td><td>IMP</td><td>first-private</td></tr>
<tr><td rowspan="2"><code>default</code> clause is not present</td><td>In a <code>parallel</code> construct</td><td>IMP</td><td>shared</td></tr>
<tr><td>In a <code>task</code>, <code>taskloop</code>, <code>target</code>, <code>target enter data</code>, <code>target exit data</code>, <code>target update</code> construct</td><td>IMP</td><td>first-private</td></tr>
<tr><td rowspan="2"><code>default</code> clause is present</td><td>In a <code>parallel</code> and <code>teams</code> construct</td><td>IMP</td><td rowspan="2">Determined by <code>default</code> clause</td></tr>
<tr><td>In a <code>task</code>, <code>taskloop</code>, <code>target</code>, <code>target enter data</code>, <code>target exit data</code>, <code>target update</code> construct</td><td>IMP</td></tr>
</table>