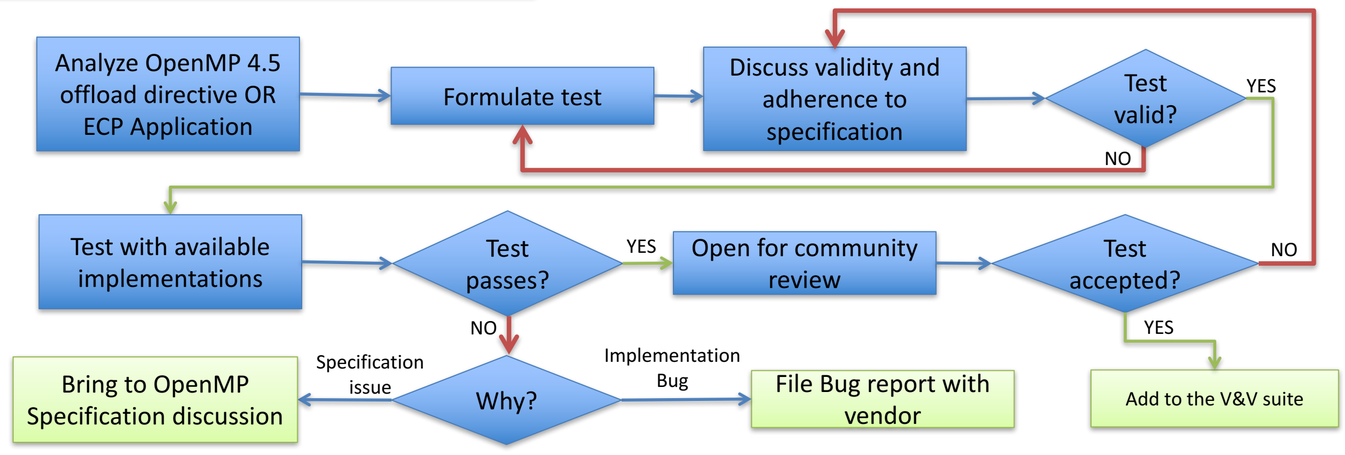
OpenMP 4.5 Validation & Verification Suite for Offloading Features

UDEL: Sergio Pino (sergiop@udel.edu), Jose Monsalve (josem@udel.edu) , Sunita Chandrasekaran (schandra@udel.edu)
ORNL: Swaroop Pophale (pophaless@ornl.gov), Oscar Hernandez (oscar@ornl.gov), David Berndholt (bernholdtde@ornl.gov)
ANL: Hal Finkel (hfinkel@anl.gov)

Abstract

- Building Validation and Verification Suite focusing on the offloading features of OpenMP 4.5 and beyond
- Creating functional and unit tests
- Collaboration with ECP application developers to create representative scientific use cases
- Project in collaboration with OpenMP community, Oak Ridge National Lab (ORNL), Argonne National Lab (ANL), University of Delaware (UDEL), IBM, LLVM, Cray, GCC
- Project uncovers compiler and runtime implementation bugs and ambiguities in the OpenMP 4.5 specification
- Compilers that this project use for evaluation include Clang/LLVM, GNU GCC, IBM XL, Cray CCE
- Target platforms include ORNL's TITAN and SummitDev (representative exascale system)
- Project is open for collaboration
- Feedback and suggestions from the HPC community is welcome - Contact any of the authors

Methodology



There are three possible positive outcomes of the process we have adopted. Either a test passes through all the checks and makes it to the validation suite, uncovers a bug in the vendor implementation of the OpenMP 4.5 standard, or highlights a contentious concept or text that is easy to misinterpret and brings it to the attention of the OpenMP community and specification developers. All tests are written agnostic to where they are executed (host vs. target). After a test executes the output indicates if the test passed or failed and where it was executed (host or target).

OpenMP offloading

Host centric execution of code: Offloading directives provides the compiler with hints to create device executable code, as well as inline all the necessary calls for device initialization, code execution and data movement between host and device.
OpenMP frees the programmer from bookkeeping data allocation and movement, as well as separate compilation of code for host and device.
OpenMP 4.5 in particular provides more control to the programmer to handle data movement between host and device.

Device code generation
Conditional exec on device
Data movement
Runtime device selection

```
#pragma omp target map(tofrom: myVar) if(myCondition) device(2)
{
   myVar ++;
}
```

Simple Test Cases

```
1  int test_map_device() {
2    int num_dev = omp_get_num_devices(), sum[num_dev], errors = 0;
3    int* h_matrix = (int*) malloc(num_dev*N*sizeof(int));
4
5    for (int dev = 0; dev < num_dev; ++dev) {
6  #pragma omp target data map(from: h_matrix[dev*N:N]) device(dev)
7      {
8  #pragma omp target map(from: h_matrix[dev*N:N]) device(dev)
9        {
10         for (int i = 0; i < N; ++i)
11           h_matrix[dev*N + i] = dev;
12       } // end target
13     } // end target data
14   }
15
16   // checking results
17   errors = 0;
18   for (int dev = 0; dev < num_dev; ++dev) {
19     sum[dev] = h_matrix[dev*N + 0];
20     for (int i = 1; i < N; ++i)
21       sum[dev] += h_matrix[dev*N + i];
22     errors |= (dev * N != sum[dev]);
23   }
24
25   return errors;
26 }
```

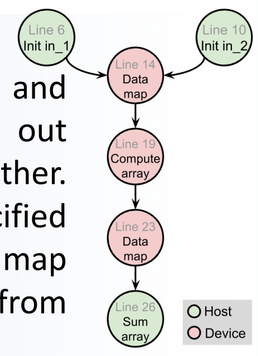
Offloading Multiple devices:
Distribute each row of a matrix to one of the available devices(lines 5-14). Each iteration performs data movement and computation in a different device. Target data region maps a portion of the matrix to each device(line 6). Computation is performed on the target region (line 8).

```
1  class B {
2  public:
3    static double VAR;
4    B() {}
5    static void modify(double* res) {
6  #pragma omp target map(tofrom: res[0:1])
7      {
8        *res = B::VAR;
9      }
10   }
11 };
12 double B::VAR = 1.0;
13
14 int test_static() {
15   int errors = 0;
16   double exp = 1.0, res = 0.0;
17
18   B::modify(&res);
19
20   errors = res != exp;
21   return errors;
22 }
```

Mapping static attribute of a class:
The unique value of the stativ VAR is mapped inside the method of a class with a target region and the map clause (lines 6-9). An OpenMP 4.5 capable compiler should capture the static variable (VAR) and map it to and from the device.

```
1  double sum = 0.0;
2  double* h_array = (double *) malloc(N * sizeof(double));
3  double* in_1 = (double *) malloc(N * sizeof(double));
4  double* in_2 = (double *) malloc(N * sizeof(double));
5
6  #pragma omp task depend(out: in_1) shared(in_1)
7  { for (int i = 0; i < N; ++i)
8      in_1[i] = 1; }
9
10 #pragma omp task depend(out: in_2) shared(in_2)
11 { for (int i = 0; i < N; ++i)
12     in_2[i] = 2; }
13
14 #pragma omp target enter data nowait \
15   map(alloc: h_array[0:N]) map(to: in_1[0:N]) \
16   map(to: in_2[0:N]) depend(out: h_array) \
17   depend(in: in_1) depend(in: in_2)
18
19 #pragma omp target nowait depend(inout: h_array)
20 { for (int i = 0; i < N; ++i)
21     h_array[i] = in_1[i]*in_2[i]; }
22
23 #pragma omp target exit data nowait \
24   map(from: h_array[0:N]) depend(inout: h_array)
25
26 #pragma omp task depend(in: h_array) \
27   shared(sum, h_array)
28 { for (int i = 0; i < N; ++i)
29     sum += h_array[i]; }
30
31 #pragma omp taskwait
32 errors = 2.0*N != sum;
```

Task dependencies:
Task graph composed of host and target tasks that have in and out dependencies between each other. Asynchronous behavior is specified using the *nowait* clause. Data map tasks are separated from computation tasks.



Complex Test Cases

```
1  typedef struct node {
2      double data;
3      struct node *next;
4  } node_t;
5
6  void map_ll(node_t *head) {
7      if (!head) return;
8  #pragma omp target enter data map(to:head[:1])
9      while(head->next) {
10         // Note: explicit attachment
11         node_t *cur = head->next;
12 #pragma omp target enter data map(to:cur[:1])
13 #pragma omp target
14         {
15             head->next = cur;
16         }
17     }
18 }
19
20 void unmap_ll(node_t *head) {
21     if (!head) return;
22 #pragma omp target exit data map(from:head[0].data)
23     while(head->next) {
24         // Note: only copies back the data element to avoid
25         overwriting next pointer
26 #pragma omp target exit data map(from:head[0].next[0].data)
27     }
}
```

Mapping Linked list to device:
The map_ll function (line 5) uses *target enter data* directive to first map the head of the linked list, and then map the pointer to the next link using array dereferencing syntax. The unmap_ll (line 19) function explicitly copies the data using map-type from with target exit data map.

```
1  #define RealType double
2  #define N 10
3
4  #pragma omp declare target
5  class MyVector
6  {
7  public:
8
9      inline RealType operator()(int i, int j, int k) const
10     {
11         return X[k+Length[2]*(j+Length[1]*i)];
12     }
13
14     inline RealType& operator()(int i, int j, int k)
15     {
16         return X[k+Length[2]*(j+Length[1]*i)];
17     }
18
19     MyVector(int l, int m, int n)
20     {
21         Length[0] = l;
22         Length[1] = m;
23         Length[2] = n;
24         X = new RealType[l*m*n];
25     }
26
27     RealType *& getData() { return X; }
28
29     RealType * getData() const { return X; }
30
31     int getSize() const { return Length[0]*Length[1]*Length[2]; }
32
33     int Length[3];
34     RealType * X;
35 };
36 #pragma omp end declare target
37
38 int main () {
39
40     MyVector gamma (N, N, N);
41     int size = gamma.getSize();
42
43 #pragma omp target enter data map(to:gamma)
44 #pragma omp target enter data map(to:gamma.X[0:size]) map(to:gamma
       .Length)
45
46 #pragma omp target
47     for(int i = 0 ; i < N ; i++)
48         for(int j = 0 ; j < N ; j++)
49             for(int k = 0 ; k < N ; k++)
50                 gamma(i,j,k) = 1.0;
51
52 #pragma omp target exit data map(from:gamma.X[0:size])
53
54     for(int i = 0 ; i < N ; i++)
55         for(int j = 0 ; j < N ; j++)
56             for(int k = 0 ; k < N ; k++)
57                 cout << gamma(i,j,k) << ". ";
58     return 0;
59 }
```

Deep Copy of Classes:
This code came from analyzing a full scale ECP application. It uses the *declare target* directive (line 4 to 36) to ensures that procedures and global variables can be executed and data can be accessed on the device. When the C++ methods are encountered, device-specific versions of the routines are created that can be called from a target region.
Deep copy is performed through the use of target enter data (lines 43 and 44) by first mapping the class and then the individual class members. Computation is performed on the device (line 46). After computation is over, the data is copy back to the host (line 52).

Current Snapshot

Our intention is to develop a test suite for the entire **OpenMP 4.5** specification. We divide our tests by directives. These tests have gone through the methodology described previously. Some of them have result in different bug reports.
Following is a snapshot of the current suite

- Tested using 4 different compilers: Clang, IBM XL, GCC and Cray CCE.
- Target platforms:
  - **Titan Cray XK7:** AMD Opteron x64 + Nvidia K20X
  - **SummitDev:** IBM Power8 + NVIDIA TESLA P100

		Platform: Summitdev	Platform: Summitdev	Platform: Titan	Platform: Summitdev
Test	Lang	Compiler: Clang 3.8.0 ORNL Version	Compiler: IBM XL 13.1.6	Compiler: CCE 8.6.3	Compiler: GCC 7.1.1
offloading_success.c	C	PASS	PASS	PASS	PASS
offloading_success.cpp	C++	PASS	PASS	PASS	PASS
target/test_target_is_device_ptr.c	C	PASS	PASS	PASS	PASS
target/test_target_map_array_default.c	C	PASS	PASS	PASS	PASS
target/test_target_map_pointer.c	C	PASS	PASS	PASS	PASS
target/test_target_map_pointer_default.c	C	PASS	PASS	PASS	PASS
target/test_target_map_scalar_default.c	C	PASS	PASS	PASS	PASS
target/test_target_if.c	C	PASS	PASS	PASS	PASS
target/test_target_map_global_arrays.c	C	PASS	PASS	PASS	PASS
target/test_target_map_local_array.c	C	PASS	PASS	PASS	PASS
target_data/test_target_data_if.c	C	PASS	PASS	PASS	PASS
target_data/test_target_data_map.c	C	PASS	PASS	PASS	PASS
target_enter_exit_data/test_target_enter_exit_data_async.c	C	PASS	PASS	PASS	PASS
target_enter_exit_data/test_target_enter_exit_data_map_global_array.c	C	PASS	PASS	PASS	PASS
target_enter_exit_data/test_target_enter_exit_data_map_malloced_array.c	C	PASS	PASS	PASS	PASS
target/test_target_map_struct_default.c	C	PASS	PASS	C. FAIL	PASS
target/test_target_device.c	C	PASS	PASS	C. FAIL	PASS
target/test_target_async.c	C	PASS	R. FAIL	PASS	PASS
target_data/test_target_data_map_array_sections.c	C	PASS	PASS	R. FAIL	PASS
target_data/test_target_data_map_classes.cpp	C++	PASS	PASS	C. FAIL	PASS
target_data/test_target_data_map_devices.c	C	PASS	PASS	C. FAIL	PASS
target_enter_exit_data/test_target_enter_exit_data_devices.c	C	PASS	PASS	C. FAIL	PASS
target_enter_exit_data/test_target_enter_exit_data_struct.c	C	PASS	PASS	C. FAIL	PASS
application_kernels/linked_list.c	C	PASS	PASS	C. FAIL	PASS
target_data/test_target_data_use_device_ptr.c	C	PASS	PASS	R. FAIL	C. FAIL
target/test_target_defaultmap.c	C	PASS	R. FAIL	R. FAIL	PASS
target/test_target_map_classes_default.cpp	C++	R. FAIL	R. FAIL	C. FAIL	C. FAIL
target_enter_exit_data/test_target_enter_exit_data_classes.cpp	C++	C. FAIL	R. FAIL	C. FAIL	C. FAIL