OpenMP 4.5 Implementations:

Evaluation & Verification Of Offloading Features

University of Delaware, Computer and Information Sciences

UDEL: 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)

Visit our website

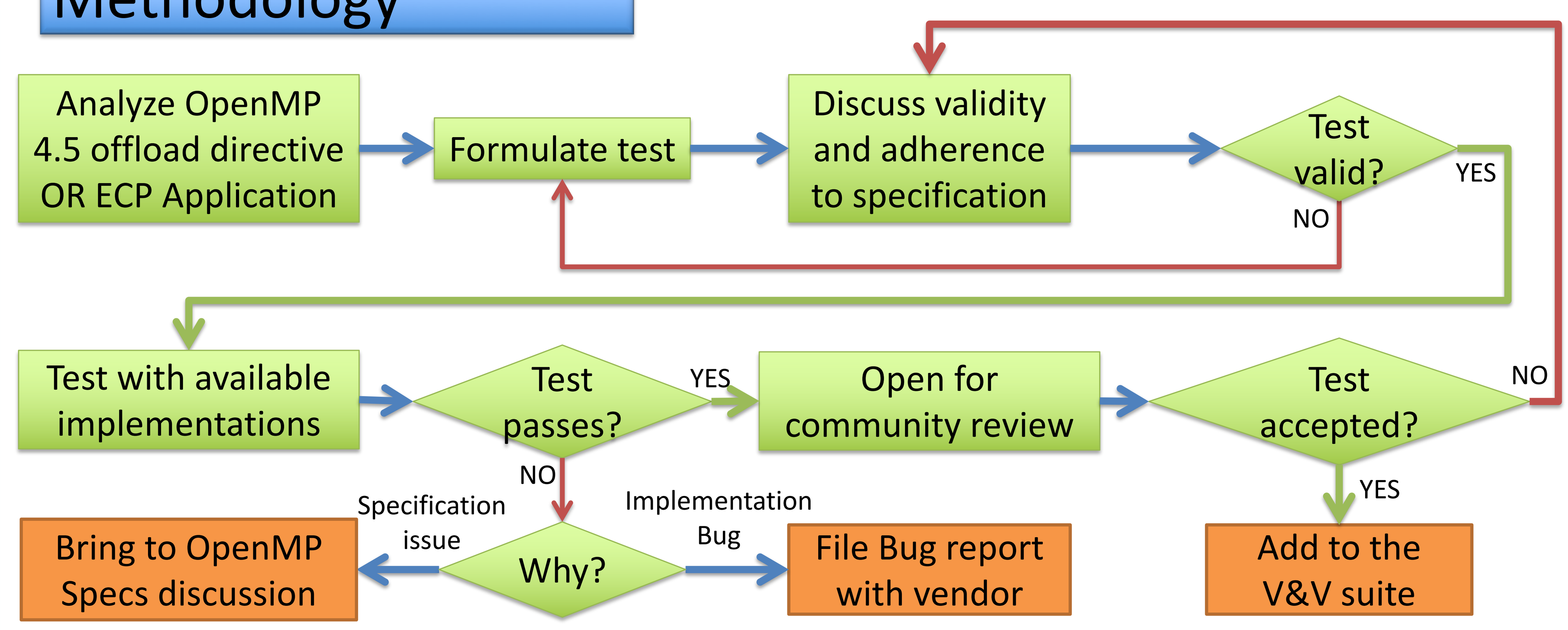
Abstract

On the quest of High Performance Computing systems capable of achieving high parallelism on scientific applications, a trend of adding acceleration devices with specialized hardware architectures can be observed. An evidence of this are the 102 supercomputers of the latest top 500 list that use an accelerator or co-processor of which 86 use GPGPUs. Furthermore, new programming languages and frameworks have been created to support and program accelerators. While each vendor may offer its own programming solution, traditional programming frameworks such as OpenMP works towards solving this issue by adapting its standard starting at version 4.0 to support target devices. Although currently it is not the only option, all solutions share a common objective of increasing code portability across vendors as well as multiple architectural generations.

However, maturity and correctness of these solution needs to be tested. In particular for OpenMP, there is a breach between the moment the standard is released and when compilers and vendors implement support for such standard. Hence, there is a need of a methodology to assess the quality of an implementation of the OpenMP standard, as well as evaluate system compliance during system deployments. Such methodology should reveal possible bugs or conflicts on the interpretation of the standard, as well as provide metrics to characterize the quality level of many implementation.

To this end, this work presents a tests suite for OpenMP 4.5 that is currently under development. This suite follows a well thought methodology that was stablished in advanced to provide a common and steady ground, allowing vendors, systems developers and programmers to asses the level of readiness and maturity of an OpenMP 4.5 implementation for a particular vendor.

Methodology



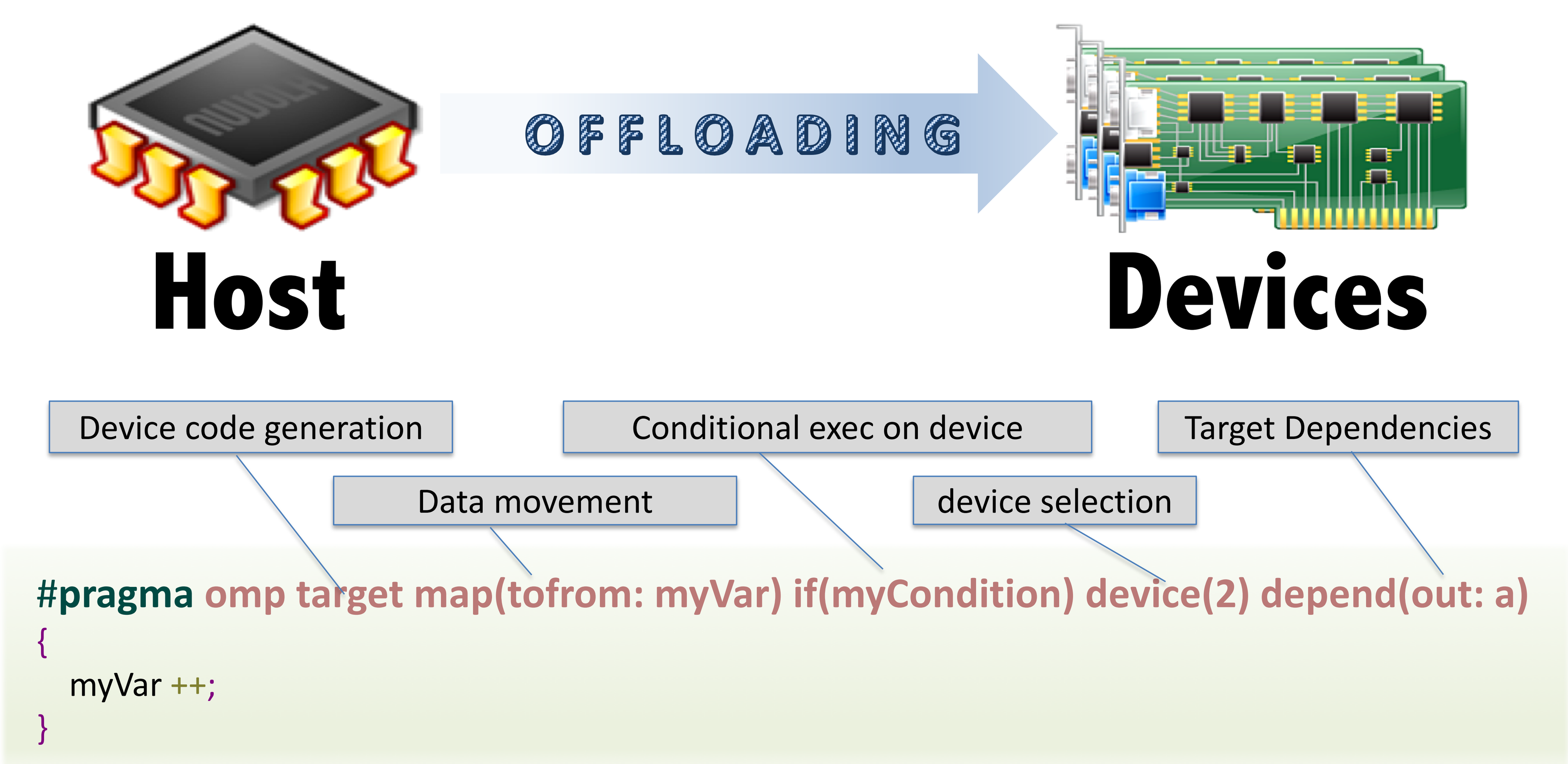
There are three possible positive outcomes of our methodology to build tests. Either a test passes through all the checks and makes it to the validation suite, or it uncovers a bug in the vendor implementation of the OpenMP 4.5 standard, or it highlights a contentious concept or text that is easy to misinterpret which needs to be brought 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

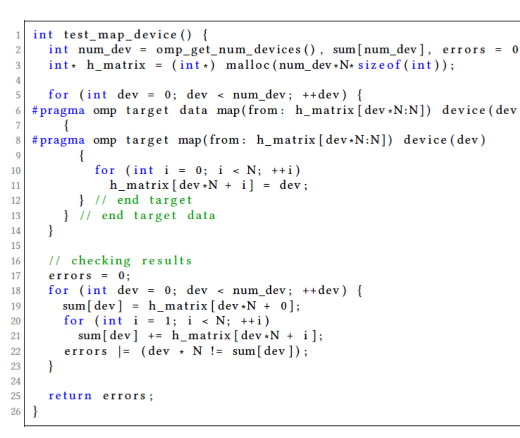
Host centric execution of code: Offloading directives provides the compiler with hints to create device executable code, as well as inline all the necessary calls for device initialization, code execution and data movement between host and device.

OpenMP frees the programmer from bookkeeping data allocation and movement, as well as separate compilation of code for host and device.

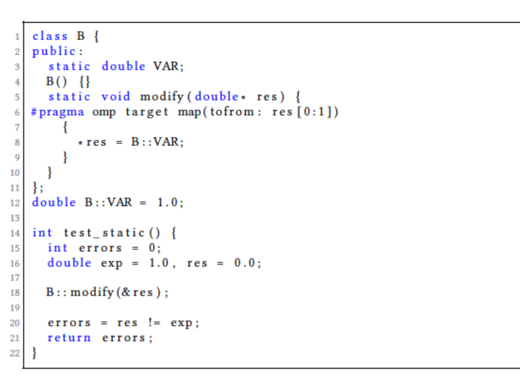
OpenMP 4.5 in particular provides more control to the programmer to handle data movement between host and device.



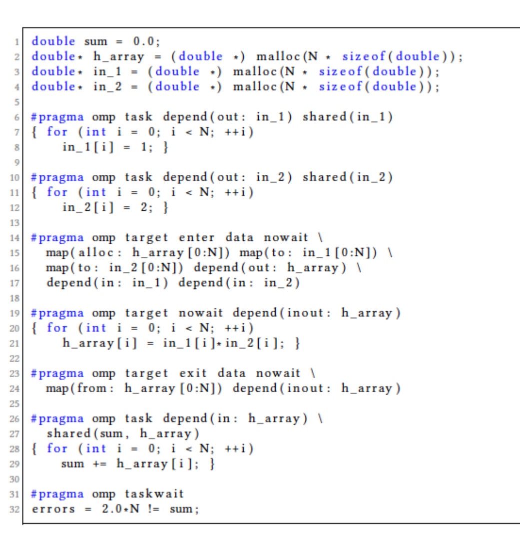
Simple Test Cases



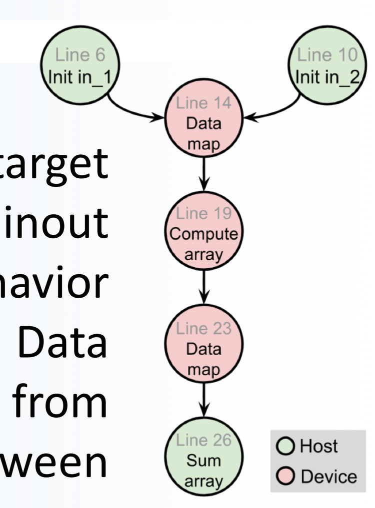
Offloading Multiple devices:
Distribute each row of a matrix to one of the available devices(lines 5-14). Each iteration performs data movement and computation in a different device. Target data region maps a portion of the matrix to each device(line 6). Computation is performed on the target region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



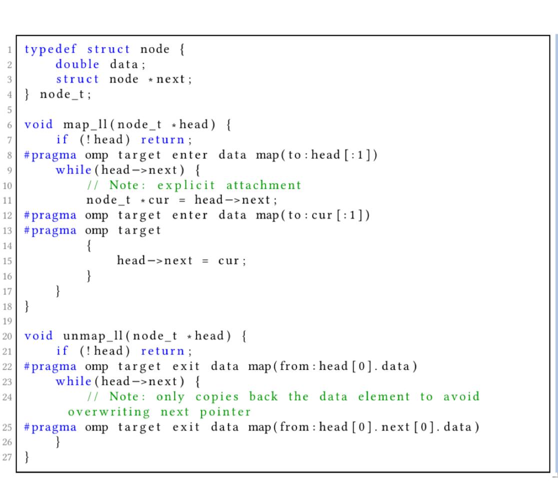
Mapping static attribute of a class:
The unique value of the stativ VAR is mapped inside the method of a class with a target region and the map clause (lines 6-9). An OpenMP 4.5 capable compiler should capture the static variable (VAR) and map it to and from the device.



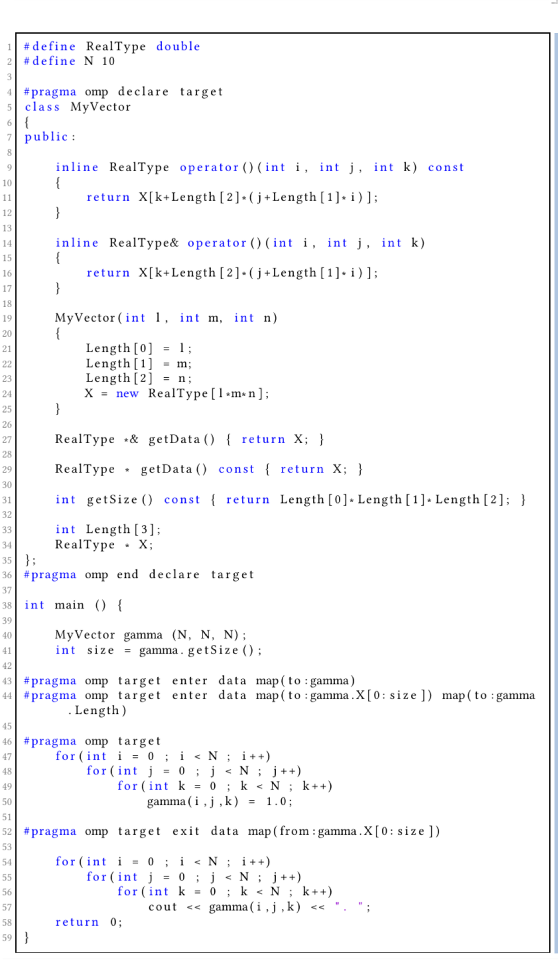
Task dependencies:
Task graph made out of host and target tasks with in, out and inout dependencies. Asynchronous behavior specified using the *nowait* clause. Data mapping tasks are separated from computation tasks. Signals between host and target in both directions



Complex Test Cases



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.



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 classify our tests by directives and clauses. The aforementioned methodology was used. Some of them have resulted in multiple 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

Oak Ridge National Laboratory

ECP Exascale Computing Project