OpenMP 4.5 Implementations: Evaluation & Verification Of Offloading Features

UDEL: Jose Montaño (jose@udel.edu) - Srinu Chandrasaran (srinu@udel.edu)
ORNL: Swapno Pophalo (pophalo@ornl.gov), Oscar Hernandez (oscar@ornl.gov), David Berndt (berndt@ornl.gov)
ANL: Hal Finkel (hfinkel@anl.gov)

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 GPUs. 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 solutions need to be tested. In particular for OpenMP, there is a reach between the moment the standard is released and when compliers 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 stabalised in advanced to provide a common and steady ground, allowing vendors, systems developers and programmers to assess the level of readiness and maturity of an OpenMP 4.5 implementation for a particular vendor.

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.

Methodology

- **Analyse OpenMP 4.5 offload directive OR ECP Application**
- **Formulate test**
- **Discuss validity and adherence to specification**
- **Test valid?**
  - YES
  - NO
- **Test with available implementations**
- **Test passes?**
  - YES
  - NO
- **Open for community review**
- **Test accepted?**
  - YES
  - NO
- **File Bug report with vendor**
- **Add to the V&V suite**

Simple Test Cases

**Olloffing 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 static VAR is mapped inside the defined 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 is specified using the nowait clause. Data mapping tasks are separated from computation tasks. Signals between host and target in both directions

Complex Test Cases

```c
#pragma omp target map(tofrom: myVar) if(myCondition) device(2) depend(out: a)
{
    myVar += a;
}
```

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