7 What is Next?

As this book is being printed, the OpenMP language committee is actively working on the next version of the specifications. Although the release number is subject to change, it probably will be called “OpenMP 5.0.”

Some of the future directions of OpenMP may be determined from the Technical Reports that are being released by the Language Committee. These can be found at [21]. Until these proposals for extending OpenMP have passed the user feedback process, and have been approved by the OpenMP Architecture Review Board (ARB), nothing is final, but these are the topics and features most likely included:

- Memory model.
- Support for Fortran 2003 and the latest C/C++11 standards.
- Interoperability with other parallel programming models.
- APIs to enable better debug and profiling tools.
- The `depend` clause on the `taskwait` construct.
- Affinity support for tasks.
- An environment variable to display affinity information at runtime.
- Support for reductions with tasks.
- Enhancements to task dependences.
- Memory management.
- Heterogeneous architectures.

The last two topics, memory management and heterogeneous architectures, are discussed in more detail in the following sections.

7.1 Memory Management

The memory subsystems of modern computing devices have a deep hierarchy, employing a variety of memory technologies, including main memory, high-bandwidth memory, texture memory, scratchpad memory, and multiple levels of cache. The hierarchy found in future systems is expected to become even deeper, following the introduction of non-volatile RAM and other developments. The particular challenge for the user is that each of these technologies has a different programming interface, and distinct performance characteristics. Addressing these differences becomes increasingly critical for all types of computing where performance is tied to memory bandwidth.

Under the term *Memory Management*, the OpenMP Language Committee is currently working on concepts and constructs to support the use of these memory types from within OpenMP. In January 2017, the OpenMP Technical Report 5 – Memory Management Support for OpenMP 5.0 – was released [21], describing the current approach to this topic. A detailed description is provided in [23]. These are the three new concepts proposed in this paper:

- Memory spaces and allocators - A *memory space* refers to a memory resource available in the system, at the time the OpenMP program is executed. Memory spaces are differentiated by their characteristics, which are expressed by *traits*. An *allocator* is an object that allocates and frees memory from an associated memory space.
- Memory allocation API - The `omp_alloc()` and `omp_free()` API routines are provided for C/C++ to allocate and deallocate memory using an allocator.
- The `allocate` directive and clause - The new `allocate` directive and clause allow for the allocation of variables, without the explicit use of the aforementioned API and, in consequence, support Fortran. Both offer several modifiers to impact their behavior.

Memory management and affinity are related. While OpenMP 4.0 introduced the thread affinity model, the concept of memory management may build the foundation to provide support for data affinity in OpenMP. Data affinity refers to the association of data with computation. Managing data affinity to threads is possible by leveraging the functionality provided by the Operating System. The best approach to provide data affinity to tasks in OpenMP is still an open (research) question, however. This is of high interest, because task scheduling remains opaque and may lead to performance variations on NUMA architectures.

Given the complexity of this topic, it remains to be seen how far, and in which form, these concepts find their way into OpenMP 5.0, and subsequent releases.

7.2 Heterogeneous Architectures

Accelerators and general-purpose processors are becoming more tightly coupled. In particular in the memory subsystem, there is a trend toward a single, virtual memory address space among all devices in a compute node. What this means for OpenMP is that mapped variables might evolve to be more akin to shared variables, and it will become easier to map (or share) complex data structures across devices.

In some ways, general-purpose and accelerator processors are becoming more similar. For example, new generations of both types have wider SIMD instructions, and more cores, and thus more available threads. Both types of processors might ultimately converge back into one. There might be subsets of processors in a node that are more tightly coupled and the device constructs could be used to target code and data to such processor domains.

The work improve the support in OpenMP for heterogeneous architectures is ongoing. Some of the improvements that have been worked on are:

- Implicitly map functions - Remove the requirement that a function called from a target region, or another mapped function, must appear in a `declare target` directive. The compiler can infer that a function is mapped when it is called from a target region, or another mapped function.

- Provide better C++ support - Allow C++ class variables to be mapped when the class includes static members, or virtual member functions. The first step is to allow the variable to be mapped. It remains an open question as to whether an implementation may support invoking a virtual member function of a mapped class variable on the accelerator.

- Map complex data structures - Map complex data structures, such as linked lists, arrays of pointers, or structures and classes with pointer members. On heterogeneous systems, where the host and accelerator do not share the same address space, copying complex data structures requires systematically de-referencing pointers, which is a process referred to as deep-copy.

- Add device types - New syntax enables conditionally selecting the clauses on a construct for a specific device type. New routines and clauses are able to target any device of a certain type.

- Workshare loops across devices - Distribute the iterations of a loop across either multiple accelerators, or the host device, plus one or more accelerators.

- Support for device-specific environment variables - Extend the syntax of OpenMP environment variables to indicate that their values are applied to devices other than the host. The method of initializing ICVs with environment variables is currently implementation-defined for accelerator devices.

Given the popularity of heterogeneous architectures across all types of computing, it is expected that, in some form, many of these new features will be included in OpenMP 5.0, or a later release.