0.33 pages - history of OpenACC and OpenMP, and future of OpenACC and OpenMP

1 page - overview of OpenACC directives; overview of OpenMP 4 device directives

History

In the 1980's, SMP vendors typically defined directives for specifying how their SMP's divided up work on their individual processors. In the late 90's, the OpenMP Architecture Review Board was formed to develop a specification that could be used commonly across multiple SMP's. The first OpenMP specification was released for Fortran in 1997, and for C/C++ in 1998. In version 3.0 in 2008, task parallelism support was added and notably, in version 4.0 in 2013, accelerator support was added.

OpenACC's history is much shorter. It was originally developed by PGI, Cray, and NVIDIA, with version 1.0 being released in 2011. Version 2.0 was released in 2013, adding useful features such as routine parallelization and nested parallelization (enabled by new CUDA capabilities), and other useful features as API routines.

Directives Overview

OpenMP's fundamental construct is the parallel construct. When a parallel construct is encountered by a thread, that thread creates a team of new threads, becoming that team's master. The threads are assigned numbers, with the master given number 0.The threads each execute a copy of the parallel region's code. None of the child threads may continue execution beyond the parallel region, and if any thread terminates within the region, the work done up to that point is undefined. Nested parallelism is legal in OpenMP, so parallel constructs may contain nested parallel constructs.

Inside a parallel region, a for construct may be specified. A for region is associated with a for loop, and upon encountering the region, a different thread from the current team is assigned to each iteration of the for loop. A parallel for region is also defined as a shorthand for a parallel region containing nothing but a single for region. OpenMP 4.0 adds a similar construct, simd, which divides work of the associated loop up among SIMD lanes of the current thread.

Other workshare constructs are specified as well. The sections construct allows the programmer to divide up a region into sections, each of which will be executed by a thread in the team. The single construct specifies that only one thread is to execute the given region. The workshare construct divides a block into multiple units of work shared amongst the thread team. Only certain constructs and statements may exist within the given block.

The task construct was introduced in version 3.0 as part of OpenMP's new task parallelism support. When a thread encounters a task construct, a task is generated and assigned to some thread in the current team. Tasks can be executed then in arbitrary order. OpenMP also specifies severalother constructs, such as taskyield, taskwait, and taskgroup, to control how and when tasks are executed.

MENTION TARGET CLAUSE?

OpenACC has significantly fewer constructs. This is in part because OpenACC is specifically for use with massively parallel accelerator hardware, where OpenMP tries to suppport accelerators as well as multicore CPU. As a result, task parallelism is not present in OpenACC. However, many of OpenACC's construct are analogous to OpenMP ones.

OpenACC also has a parallel construct. A parallel construct, when encountered, creates gangs of workers to execute the region on the accelerator. A loop construct is also available, specifying the type of parallelism for the associated loop. There is also the parallel loop abbreviation, a shorthand for a loop construct immediately inside a parallel construct.

The kernels construct specifies a region that is to be divided into a sequence of kernels for accelerator execution. The loop and shorthand kernels loop are available for kernels directives as they are for parallel directives.

A data directive is also defined. It specifies for a region what data should be copied into the accelerator upon entry to the region and copied out on exit and defines what data is to be allocated on the device for the duration of the region. Alternatively, the specification provides enter data and exit data directives that specify data allocation in the same way, but without defining a region - enter data device allocations simply exist until exit data directives deallocate them.