The OpenACC™ API
QUICK REFERENCE GUIDE

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++, and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.
Kernels Construct
An accelerator kernels construct surrounds loops to be executed on the accelerator, typically as a sequence of kernel operations.
C
#pragma acc kernels {clause [[,] clause]...} new-line
(structured block)
FORTRAN
!$acc kernels {clause [[,] clause]...} new-line
(structured block)
!$acc end kernels
An data clause is allowed.
OTHER CLAUSES
if( condition )
When the condition is nonzero or true, the kernels region will execute on the accelerator; otherwise, it will execute on the host.
async( expression )
The kernels region executes asynchronously with the host.

Data Construct
An accelerator data construct defines a region of the program within which data is accessible by the accelerator.
C
#pragma acc data {clause [[,] clause]...} new-line
(structured block)
FORTRAN
!$acc data {clause [[,] clause]...} new-line
(structured block)
!$acc end data
An any data clause is allowed.
OTHER CLAUSES
if( condition )
When the condition is nonzero or true, no data will be allocated or moved to or from the accelerator.
async( expression )
Data movement between the host and accelerator will occur asynchronously with the host.
Data Clauses
The description applies to the clauses used on parallel constructs, kernels constructs, data constructs, declare constructs, and executable update directives.
copy( list )
Allocates the data in list on the accelerator and copies the data from the host to the accelerator when entering the region, and copies the data from the accelerator to the host when exiting the region.
copyin( list )
Allocates the data in list on the accelerator and copies the data from the host to the accelerator when entering the region.
copyout( list )
Allocates the data in list on the accelerator and copies the data from the accelerator to the host when exiting the region.
create( list )
Allocates the data in list on the accelerator, but does not copy data between the host and device.
present( list )
The data in list must be already present on the accelerator, from some containing data region; that accelerator copy is found and used.
copy_in( list )
If the data in list is already present on the accelerator from some containing data region, that accelerator copy is used; if it is not present, this behaves like the copy clause.
copy_out( list )
If the data in list is already present on the accelerator from some containing data region, that accelerator copy is used; if it is not present, this behaves like the copyin clause.
create_in( list )
If the data in list is already present on the accelerator from some containing data region, that accelerator copy is used; if it is not present, this behaves like the copyin clause.
create_out( list )
If the data in list is already present on the accelerator from some containing data region, that accelerator copy is used; if it is not present, this behaves like the copyout clause.
create( list )
If the data in list is already present on the accelerator from some containing data region, that accelerator copy is used; if it is not present, this behaves like the create clause.
deviceptr( list )
C and C++; the entries must be pointer variables that contain device addresses, such as from acc_malloc.
Fortran: the list entries must be dummy arguments, and may not have the pointer, allocatable or value attributes.

Host Data Construct
A host_data construct makes the address of device data available on the host.
C
#pragma acc host_data {clause [[,] clause]...} new-line
(structured block)
FORTRAN
!$acc host_data {clause [[,] clause]...} new-line
(structured block)
!$acc end host_data
CLAUSES
use_device( list )
Directs the compiler to use the device address of any entry in list, for instance, when assigning a variable to procedure.
Loop Construct
A loop construct applies to the immediately following loop or nested loops, and describes the type of accelerator parallelism to use to execute the iterations of the loop.
C
#pragma acc loop {clause [[,] clause]...} new-line
FORTRAN
!$acc loop {clause [[,] clause]...} new-line
CLASSES
collapse( n )
Applies this directive to the following n nested loops.
seq
Executes this loop sequentially on the accelerator.
private( list )
A copy of each variable in list is created for each iteration of the loop.
reduction( operator:var )
See reduction clause for parallel construct.
Clauses unique to an OpenACC parallel region
gang
Shares the iterations of this loop across the gangs of the parallel region.
worker
Shares the iterations of this loop across the workers of the gang.
vector
Executes the iterations of this loop in SIMD or vector mode.

Cache Construct
A cache construct may be added at the top of a loop. The elements or subarrays in the list are cached in the software-managed data cache.
C
#pragma acc cachefrom( list )
FORTRAN
!$acc cachefrom( list )
Update Directive
The update directive copies data between the host memory and data allocated in device memory or vice versa. An update directive may appear in any data region, including an implicit data region.
C
#pragma acc update {clause [[,] clause]...} new-line
FORTRAN
!$acc update {clause [[,] clause]...} new-line
CLASSES
host( list )
Copies the data from the accelerator to the host.
device( list )
Copies the data from the host to the accelerator.
if( condition )
When the condition is nonzero or true, no data will be moved to or from the accelerator.
async( expression )
Data movement between the host and accelerator will occur asynchronously with the host; the expression value may be used in a wait directive or API call.