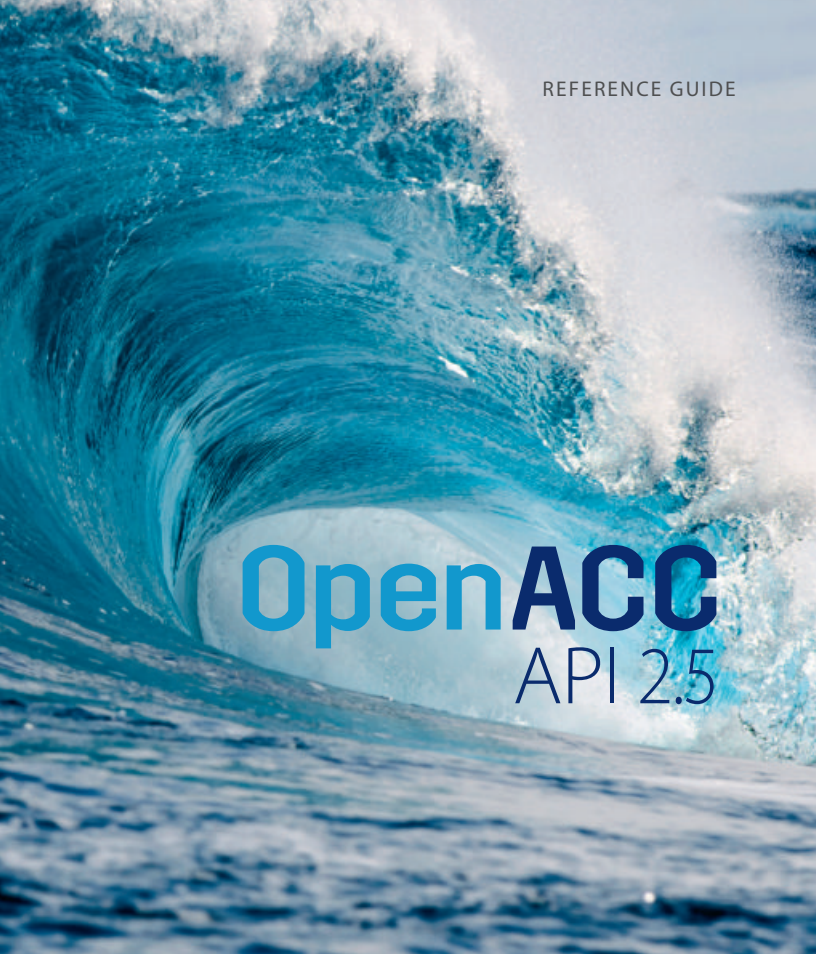

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator device, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C and C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

General Syntax

C/C++
#pragma acc *directive [clause [[,] clause]...] new-line*

FORTRAN

!$acc *directive [clause [[,] clause]...]*

An OpenACC construct is an OpenACC directive and, if applicable, the immediately following statement, loop or structured block.

Parallel Construct

A **parallel** construct launches a number of gangs executing in parallel, where each gang may support multiple workers, each with vector or SIMD operations.

C/C++

#pragma acc parallel *[clause [[,] clause]…] new-line*
{ structured block }

FORTRAN

!$acc parallel *[clause [[,] clause]…]*
structured block
!$acc end parallel

Compute Construct and Data clauses are also allowed; data clauses on the parallel construct modify the structured reference counts for the associated data.

OTHER CLAUSES

reduction(*operator: list* **)**
A private copy of each variable in ***list*** is allocated for each gang. The values for all gangs are combined with the ***operator*** at the end of the parallel region. Valid C and C++ operators are **+, *, max**, **min**, **&**, **|**, **^**, **&&**, **||**. Valid Fortran operators are **+**, *****, **max**, **min**, **iand**, **ior**, **ieor**, **.and.**, **.or.**, **.eqv.**, **.neqv.**

private(*list* **)**
A copy of each variable in ***list*** is allocated for each gang.

firstprivate(*list* **)**
A copy of each variable in ***list*** is allocated for each gang and initialized with the value of the variable of the encountering thread.

Kernels Construct

A **kernels** construct surrounds loops to be executed on the device, typically as a sequence of kernel operations.

C/C++

#pragma acc kernels *[clause [[,] clause]…] new-line*
{ structured block }

FORTRAN

!$acc kernels *[clause [[,] clause]…]*
structured block
!$acc end kernels

Compute Construct and Data clauses are also allowed; data clauses on the kernels construct modifty the structured reference counts for the associated data.

Compute Construct Clauses

`if(` *condition* **`)`**
When the ***condition*** is nonzero or .TRUE. the kernels region will execute on the device; otherwise, the encountering thread will execute the region.

`default( none )`
Prevents the compiler from implicitly determining data attributes for any variable used or assigned in the construct.

`default( present )`
Implicitly assume any non-scalar data not specified in a data clause is present.

`device_type` or **`dtype(`** *[* | device-type-list]* **`)`**
May be followed by any of the clauses below. Clauses following **`device_type`** will apply only when compiling for the given device type(s). Clauses following **`device_type( * )`** apply to all devices not named in another **`device_type`** clause.

`async` *[(expression)]*
The kernels region executes asynchronously with the encountering thread on the corresponding async queue.

`wait` *[(expression-list)]*
The kernels region will not begin execution until all actions on the corresponding async queue(s) are complete.

`num_gangs(` *expression* **`)`**
Controls how many parallel gangs are created.

`num_workers(` *expression* **`)`**
Controls how many workers are created in each gang.

`vector_length(` *expression* **`)`**
Controls the vector length on each worker.

Data Construct

An device **`data`** construct defines a region of the program within which data is accessible by the device.

C/C++
`#pragma acc data` *[clause[[,] clause]…] new-line*
{ structured block }

FORTRAN
`!$acc data` *[clause[[,] clause]…]*
structured block
`!$acc end data`

Data clauses are also allowed; data clauses on the data construct modify the structured reference counts for the associated data.

OTHER CLAUSES
`if(` *condition* **`)`**
When the ***condition*** is zero or .FALSE. no data will be allocated or moved to or from the device.

Enter Data Directive

An **enter data** directive is used to allocate and move data to the device memory for the remainder of the program, or until a matching **exit data** directive deallocates the data.

C/C++

#pragma acc enter data *[clause[[,] clause]…] new-line*

FORTRAN

!$acc enter data *[clause[[,] clause]…]*

CLAUSES

if(*condition* **)**

When the ***condition*** is zero or .FALSE. no data will be allocated or moved to the device.

async *[(expression)]*

The data movement executes asynchronously with the encountering thread on the corresponding async queue.

wait *[(expression-list)]*

The data movement will not begin execution until all actions on the corresponding async queue(s) are complete.

copyin(*list* **)**
create(*list* **)**

See Data Clauses; data clauses on the enter data directive modify the dynamic reference counts for the associated data.

Exit Data Directive

For data that was created with the **enter data** directive. the **exit data** directive moves data from device memory and deallocates the memory,

C/C++

#pragma acc exit data *[clause[[,] clause]…] new-line*

FORTRAN

!$acc exit data *[clause[[,] clause]…]*

CLAUSES

if(*condition* **)**

When the ***condition*** is zero or .FALSE. no data will be moved from the device or deallocated.

async *[(expression)]*

The data movement executes asynchronously with the encountering thread on the corresponding async queue.

wait *[(expression-list)]*

The data movement will not begin execution until all actions on the corresponding async queue(s) are complete.

finalize

Sets the dynamic reference count to zero.

copyout(*list* **)**
delete(*list* **)**

acc_get_default_async()
Returns the async queue used by default when no queue is specified in an async clause.

acc_set_default_async()
Sets the default async queue used by default when no queue is specified on an async clause.

acc_on_device(*devicetype* **)**
In a **parallel** or **kernels** region, this is used to take different execution paths depending on whether the program is running on a device or on the host.

acc_malloc(size_t)
Returns the address of memory allocated on the device device.

acc_free(d_void*)
Frees memory allocated by **acc_malloc.**

acc_map_data(h_void*, d_void*, size_t)
Creates a new data lifetime for the host address, using the device data in the device address, with the data length in bytes.

acc_unmap_data(h_void*)
Unmaps the data lifetime previously created for the host address by **acc_map_data**.

acc_deviceptr(h_void*)
Returns the device pointer associated with a host address. Returns NULL if the host address is not present on the device.

acc_hostptr(d_void*)
Returns the host pointer associated with a device address. Returns NULL if the device address is not associated with a host address.

```
acc_memcpy_to_device( d_void*, h_void*,
                      size_t )
acc_memcpy_to_device_async( d_void*, h_void*,
                            size_t, int )
```

Copies data from the local thread memory to the device.

```
acc_memcpy_from_device( h_void*, d_void*,
                        size_t )

acc_memcpy_from_device_async( h_void*,
                              d_void*,
                              size_t, int)
```

Copies data from the device to the local thread memory.

```
acc_memcpy_device( d_void*, d_void*, size_t )

acc_memcpy_device_async( d_void*, d_void*,
                         size_t, int)
```

Copies data from one device memory location to another.

DATA MOVEMENT ROUTINES

The following data routines are called with C prototype:

```
routine( h_void*, size_t )
```

and in Fortran with interface:

```
subroutine routine( a )
 type, dimension(:[,:]...) :: a
subroutine routine( a, len )
 type :: a
 integer :: len
```

The async versions are colled with C prototype:

```
routine_async( h_void*, size_t, int )
```

and in Fortran with interface:

```
subroutine routine_async( a, async )
 type, dimension(:[,:]...) :: a
 integer :: async
subroutine routine( a, len, async )
 type :: a
 integer :: len, async
```

`acc_copyin, acc_copyin_async`
Acts like an **`enter data`** directive with a **`copyin`** clause. Tests if the data is present, and if not allocates memory on and copies data to the current device. Increments the dynamic reference count.

`acc_create, acc_create_async`
Acts like an **`enter data`** directive with a **`create`** clause. Tests if the data is present, and if not allocates memory on the current device. Increments the dynamic reference count.

`acc_copyout, acc_copyout_async`
Acts like an **`exit data`** directive with a **`copyout`** and no **`finalize`** clause. Decrements the dynamic reference count. If both reference counts are zero, copies data from and deallocates memory on the current device.

`acc_copyout_finalize, acc_copyout_finalize_async`
Acts like an **`exit data`** directive with a **`copyout`** and **`finalize`** clause. Zeros the dynamic reference count. If both reference counts are zero, copies data from and deallocates memory on the current device.

`acc_delete, acc_delete_async`
Acts like an **`exit data`** directive with a **`delete`** and no **`finalize`** clause. Decrements the dynamic reference count. If both reference counts are zero, deallocates memory on the current device.

`acc_delete_finalize, acc_delete_finalize_async`
Acts like an **`exit data`** directive with a **`delete`** and a **`finalize`** clause. Zeros the dynamic reference count. If both reference counts are zero, deallocates memory on the current device.

acc_update_device, acc_update_device_async
Acts like an **update** directive with a **device** clause. Updates the corresponding device memory from the host memory.

acc_update_self, acc_update_self_async
Acts like an **update** directive with a **self** clause. Updates the host memory from the corresponding device memory.

acc_is_present
Tests whether the specified host data is present on the device. Returns nonzero or .TRUE. if the data is fully present on the device.

Environment Variables

ACC_DEVICE_TYPE *device*
The variable specifies the device type to which to connect. This can be overridden with a call to **acc_set_device_type**.

ACC_DEVICE_NUM *num*
The variable specifies the device number to which to connect. This can be overridden with a call to **acc_set_device_num**.

Conditional Compilation

The _OPENACC preprocessor macro is defined to have value yyyymm when compiled with OpenACC directives enabled. The version described here has value 201510.

More OpenACC resources available at
www.openacc.org