Wait Directive

The **wait** directive causes the host program to wait for completion of asynchronous accelerator activities. With no expression, it will wait for all outstanding asynchronous activities.

C

#pragma acc wait [(*expression* **)]** *new-line*

FORTRAN

!$acc wait [(*expression* **)]**

Runtime Library Routines

Prototypes or interfaces for the runtime library routines along with datatypes and enumeration types are available as follows:

C

#include "openacc.h"

FORTRAN

use openacc or **#include "openacc_lib.h"**

acc_get_num_devices(*devicetype* **)**
Returns the number of accelerator devices of the specified type.

acc_set_device_type(*devicetype* **)**
Sets the accelerator device type to use for this host thread.

acc_get_device_type()
Returns the accelerator device type that is being used by this host thread.

acc_set_device_num(*devicenum, devicetype* **)**
Sets the accelerator device number to use for this host thread..

acc_get_device_num(*devicetype* **)**
Returns the accelerator device number that is being used by this host thread.

acc_async_test(*expression* **)**
Returns nonzero or .TRUE. if all asynchronous activities with the given expression have been completed; otherwise returns zero or .FALSE.

acc_async_test_all()
Returns nonzero or .TRUE. if all asynchronous activities have been completed; otherwise returns zero or .FALSE.

acc_async_wait(*expression* **)**
Waits until all asynchronous activities with the given expression have been completed.

acc_async_wait_all()
Waits until all asynchronous activities have been completed.

acc_init(*devicetype* **)**
Initialized the runtime system and sets the accelerator device type to use for this host thread.

acc_shutdown(*devicetype* **)**
Disconnects this host thread from the accelerator device.

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

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

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

Implicit Data Region

An implicit data region is created at the start of each procedure and ends after the last executable statement.

Declare Directive

A declare directive is used to specify that data is to be allocated in device memory for the duration of the implicit data region of the subprogram.

C

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

FORTRAN

!$acc declare [*clause [[,] clause]…]*

Any data clause is allowed.

OTHER CLAUSES

acc_resident(*list* **)**
Declares that the variable in list are to be allocated on the accelerator device when this implicit data region is entered.

Environment Variables

ACC_DEVICE *device*
Specifies which device type to which to connect. This can be overridden with a call to **acc_set_device_type**.

ACC_DEVICE_NUM *num*
Specifies which 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 is 201111.

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.

CAPS

CRAY
THE SUPERCOMPUTER COMPANY

NVIDIA

PGI

Version 1.0, November 2011

General Syntax

C

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

FORTRAN

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

Except for executable and declarative directives, OpenACC directives apply to the immediately following statement, loop or structured block.

Parallel Construct

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

C

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

FORTRAN

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

Any data clause is allowed.

OTHER CLAUSES

if(*condition* **)**
When the condition is nonzero or true, the parallel region will execute on the accelerator; otherwise, it will execute on the host.

async(*expression* **)**
The parallel region executes asynchronously with the host.

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.

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 from the host.

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**.

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

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

Any 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

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

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.

present_or_copy(*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.

present_or_copyin(*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.

present_or_copyout(*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.

present_or_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 *list* 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

!$acc host_data *[clause [[,] clause]...]*
structured block
!$acc end host_data

CLAUSES

use_device(*list* **)**
Directs the compiler to use the device address of any entry in *list*, for instance, when passing 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]…]*

CLAUSES

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:list* **)**
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.

Clauses unique to an OpenACC **kernels** construct

gang [(*num_gangs* **)]**
Executes the iterations of the loop in parallel across at most *num_gangs* gangs.

worker [(*num_workers* **)]**
Executes the iterations of the loop in parallel across at most *num_workers* workers of a single gang.

vector *[(* *vector_length* **)]**
Executes the iterations of the loop in SIMD or vector mode, with a maximum *vector_length*.

independent
Specifies that the loop iterations are data-independent and can be executed in parallel, overriding compiler dependence analysis.

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 cache(*list* **)**

FORTRAN

!$acc cache(*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]…]*

CLAUSES

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.