# Tasking in OpenACC

Fortran Tasking Working Group, April 2025

OpenACC
More Science, Less Programming

# OPENACC TASKING BASICS

- OpenACC doesn’t define the concept of a “task” but does support tasking.
- OpenACC defines per-device Asynchronous Queues (FIFO)

1. Any non-negative integer to uniquely identify the queue.
2. Small number of special queues (synchronous queue, default)

- It is the user’s responsibility to wait on asynchronous work.

1. Wait without a queue, wait on all work.
2. Wait with a queue number, wait only on that queue.
3. Two or more queues can be joined with an *asynchronous wait*. 🤯

- Waiting on other devices were added late, not common and likely irrelevant to our discussion.

## EXAMPLES – STRAIGHT LINE

```
!$acc enter data create(A(:),B(:),C(:)) async
!$acc parallel loop async
do i=1,N { initialize arrays }
!$acc parallel loop async
do i=1,N { C(i) = 2*A(i) + B(i)}
!$acc exit data copyout(C(:)) delete(A(:),B(:)) async
call doStuff()
!$acc wait
```

- Straight-line task graph, enqueueing data transfers and compute kernels, freeing the CPU to do other things.
- Uses default asynchronous queue.

# EXAMPLES – FULLY TASKED

```
!$acc enter data create(A(:)) async(1)
!$acc enter data create(B(:)) async(2)
!$acc enter data create(C(:)) async(3)
!$acc parallel loop async(1)
do i=1,N { initialize A }
!$acc parallel loop async(2)
do i=1,N { initialize B }
!$acc parallel loop async(4) wait(1,2,3)
do i=1,N { C[i] = 2*A[i] + B[i];}
!$acc exit data copyout(C(:)) delete(A(:),B(:)) async(1) wait(4)
doStuff();
!$acc wait ! wait all, or could be wait(1)
```

- More complicated example, enables each array to be potentially initialized concurrently, then perform sum when all arrays are ready.

## EXAMPLES – FULLY TASKED (ALT)

```
!$acc enter data create(A(:)) async(1)
!$acc enter data create(B(:)) async(2)
!$acc enter data create(C(:)) async(3)
!$acc parallel loop async(1)
do i=1,N { initialize A }
!$acc parallel loop async(2)
do i=1,N { initialize B }
!$acc wait(1,2,3) async(4) ! Join 1, 2 & 3 into 4
!$acc parallel loop async(4)
do i=1,N { C(i) = 2*A(i) + B(i)}
!$acc exit data copyout(C(:)) delete(A(:),B(:)) async(1) wait(4)
doStuff();
!$acc wait ! wait all, or could be wait(1)
```

- More complicated example, enables each array to be potentially initialized concurrently, then perform sum when all arrays are ready.
- This one uses an asynchronous wait to join multiple queues.

# EXAMPLE – ARRAY SYNTAX

```
!Simple case
!$acc enter data create(A(:),B(:),C(:)) async
!One or more compute kernels can be created below,
!but all will go in same queue.
!$acc kernels async
A(:) = ...
B(:) = ...
C(:) = 2*A(:) + B(:)
!$acc end kernels
!$acc exit data copyout(C(:)) delete(A(:),B(:)) async
call doStuff()
!$acc wait
```

```
!Fully expose graph
!$acc enter data create(A(:)) async(1)
!$acc enter data create(B(:)) async(2)
!$acc enter data create(C(:)) async(3)
!$acc kernels async(1)
A(:) = ...
!$acc end kernels
!$acc kernels async(2)
B(:) = ...
!$acc end kernels
!$acc kernels async(4) wait(1,2,3)
C(:) = 2*A(:) + B(:)
!$acc end kernels
!$acc exit data copyout(C(:)) &
!$acc& delete(A(:),B(:)) async(1) wait(4)
doStuff();
!$acc wait
```

## MISC. OTHER THINGS

- Replayable graphs have been discussed, but today users rely on interoperability with CUDA or HIP Graphs instead. – OpenMP added this very recently.
- It’s now possible to wait on other devices, but this feels out of scope with our Fortran WG.

## TASKING IN OPENACC

**Pros**

- Conceptually very simple, users seem to really like this
- Maps trivially to CUDA, OpenCL, HIP, etc.
- Simple to implement, all dependencies can be resolved at runtime.

**Cons**

- Currently no native support for replayable graphs
- Limited concept of task locality (always enqueues to current device, can wait on any device [including host device])