

# OpenACC 2.6 Proposed Features

OpenACC-Standard.org

November, 2016

## 1 Introduction

This document summarizes features and changes being proposed for the next version of the OpenACC Application Programming Interface, tentatively titled OpenACC 2.6 and tentatively scheduled for completion by mid-2017. Most of these features were proposed or requested by OpenACC users. Each description includes an *issue number*, which is used for internal OpenACC committee discussion.

Feedback on any issue here and suggestions for improvements that don't appear here are welcome at [feedback@openacc.org](mailto:feedback@openacc.org).

## 2 Features

### 2.1 Deep Copy

*Issue 1.* Aggregate datatypes (classes and structs in C++, derived types in Fortran) with pointers to other data types are used to create dynamically-linked data structures. Moving these data structures between system and device memory is a challenge. We have been working on this issue for several years. Early in 2016, we produced a technical report proposing two solutions (<http://www.openacc.org/sites/default/files/TR-16-1.pdf>). One was a minor change to the OpenACC data clause behavior that would allow a user to perform a manual deep copy. The other was a significant change to allow for a more automatic true deep copy. We are proposing adding the first change to support manual deep copy. We will continue to investigate true deep copy, and are looking for a prototype implementation before proposing adoption.

### 2.2 Device Query Routines

*Issue 9.* Other device APIs support query routines, so the application can determine what device or devices are available and properties of this device. We are proposing to add similar functionality, with a query routine that can be called from the host to find properties about any device, and another query routine that can be called on the device to determine properties about itself.

## 2.3 Error Handling

*Issue 10.* We are proposing an error callback routine that an application can register with the OpenACC runtime, where the error routine will be invoked by the runtime whenever an OpenACC error occurs. Examples of errors that can arise from OpenACC are trying to update data when the data is not present, or when the code was not compiled to run on the current device. The common use case is a large application running on a large cluster, where one MPI rank on one node fails, but the rest of the program continues to execute. If the application could trap or be notified of the error, it could either gracefully exit, or perhaps even recover.

## 2.4 Polymorphic Routine Compilation

*Issue 18, 50.* The `acc routine` directive tells the compiler to generate device code for a procedure and the context in which a call to this procedure can be made. However, there are times when an application will want multiple versions of a procedure. A specific use case is the BLAS routine `saxpy`. An application may call a `seq` version of this in one place, but want to call a `vector` version in another place, and even a `worker` version in a third. In particular, library routines should be as flexible as possible. Since nested parallelism is so expensive, and these routines have no way to self-determine the context in which they were called, we are looking at proposing a way to have multiple execution modes (`gang`, `worker`, `vector`, `seq`) on the `acc routine` directive, with rules as how this modifies the behavior of contained loops and procedure calls inside this routine.

## 2.5 If Clause on Host Data Construct

*Issue 46.* We are proposing allowing an `if` clause on the `host_data` construct. This will allow a routine to call a procedure with with device addresses in some cases, and with host addresses otherwise, depending on context. The specific use case was a call to a GPU-aware MPI routine, where sometimes the data was present on the device, and sometimes not.

## 2.6 No Create data clause

*Issue 48.* We are proposing adding a new data clause, `no_create`, on compute constructs. If the current `create` clause acts like *present-or-create*, the `no_create` clause acts like *present-or-don't-create*. That is, if the data is present on the device, the device address will be passed to the generated kernel. If the data is not present, no error is issued and also no data is allocated or moved for this data. The use case is a parallel loop that conditionally accesses some array. The application will move the data to the device when the condition is *true*, but does not move the data when the condition is *false*. Currently, the application must make that data present on the device even when it will not be accessed.

## 2.7 Serial Compute Construct

*Issue 22.* We are proposing a `serial` compute construct. The `serial` compute construct is functionally equivalent to a `parallel num_gangs(1) num_workers(1) vector_length(1)` construct. This has some peculiar side effects. For instance, code in the construct is formally executing in *gang-redundant* mode, even though there is only one gang. This allows code in a `serial` region to call an `acc routine gang` procedure.

## 2.8 Data Clause Modifiers

*Issue 17.* We are proposing allowing modifiers on the data and cache clauses, such as `copyin(readonly:x)`. This may allow more aggressive code optimization, such as using a read-only cache for that data.

## 2.9 Array Reductions

*Issue 15.* We are proposing allowing arrays in reductions. Users have long requested array reductions. We have deferred this until now because of the potential high cost due to the high parallel ratio on current accelerator devices.

# 3 Clarifications and Corrections

## 3.1 Reductions Across Multiple Loops

*Issue 21.* We will clarify that a nested reduction should have a `reduction` clause on all the `acc loop` directives for loops that participate in the reduction. If a compiler autoparallelizes loops with reductions, the compiler must detect and implement the reduction properly.

## 3.2 Reductions as a Data Clause

*Issue 53.* We will clarify that a `reduction` clause on a compute construct will imply that the data appears in a `copy` clause. We also agree that a `reduction` clause on the outermost loop construct of a compute construct also implies that the data appears in a `copy` clause for the compute construct.

## 3.3 Fortran Bindings for all API Routines

*Issue 25, 30.* We are proposing to standardize on Fortran bindings for all the OpenACC API routines.

### 3.4 Asynchronous Data Constructs and the Present Table

*Issue 45.* We are proposing to clarify when data becomes and stops being present with asynchronous data constructs. Specifically, with `acc enter data async`, will the data be present immediately after the `enter data` directive, or will it be present only after the data movement is complete? Similarly, with `acc exit data async`, does the data become not present immediately, or only when the asynchronous data movement and deallocation is done?

### 3.5 Partially Shared Memory

*Issue 63.* We are proposing to allow devices where some of the memory is shared with the host and some is not. Current, OpenACC defines behavior for shared memory devices and non-shared memory devices. It does not define the behavior for a device where some of the address space is shared with the CPU, but not all. The use case today is NVIDIA GPUs with CUDA Unified Memory.

### 3.6 Fortran Optional Arguments in Data Clauses

*Issue 43.* We are proposing to define that when a Fortran optional argument appears in a data clause, if the argument is not *present* (in the Fortran sense), no error will occur. Also, the implementation must allow the Fortran *present* test on the device, and it must return `.false.` if the argument was not *present* on the host.

## 4 Deferred Topics

### 4.1 True Deep Copy

*Issue 1.* As mentioned above, we are deferring adoption of true deep copy into OpenACC until we see more experience with a prototype implementation

### 4.2 Multiple Devices

*Issue 2.* OpenACC supports multiple devices today with the `acc_set_device_num` API call and the `acc set` directive. Currently, this is usually done by having multiple threads, or multiple MPI processes, with each thread or process selecting a different device. However, it is inconvenient to use multiple devices in a single thread. There are systems today with more than 8 accelerator devices per node, and many future systems are being proposed and designed with multiple devices per node. Adding a clause on the various OpenACC constructs to select a device, as is done with the OpenMP `target` constructs, is difficult to manage. The problem is properly managing and optimizing the data, making sure the correct data is present on each device, and then balancing the load across devices.

### 4.3 Host as a Device

*Issue 7, 24.* Some current implementations already support OpenACC parallel execution on the host. We are also looking at treating the host just like another device. In such a mode, the system would look like it had just one more device with different execution characteristics and memory behavior. As with multiple devices, the problem will be managing the data and balancing the load. When treating the host as a device, we also must decide whether the original host thread participates in the parallel execution, as with OpenMP `parallel` constructs, or whether the compute constructs can be launched on the host asynchronously, as with accelerator devices.

### 4.4 Memory Hierarchy

*Issue 3.* The split between host and device memory has some of the characteristics of a memory hierarchy, with a large but low bandwidth system memory and a smaller but higher bandwidth device memory. Current and future systems are being designed with an exposed memory hierarchy, even without accelerators, such as the Intel Xeon Phi x200 (Knights Landing) nodes. Future systems may also include an even larger nonvolatile memory. We are exploring how to expose and exploit these features, without requiring the user to change the application for each target.

### 4.5 Reduction for C++ Complex

*Issue 6.* One missing piece for OpenACC is support for reductions for the C++ `complex` class. The problem is that from the implementation or compiler point of view, a C++ `complex` class is just a struct with multiple data elements. OpenMP addressed this problem with *user-defined reductions*. Since the only useful reduction for complex is a sum, a simpler method may be possible as well, such as treating it like an array reduction (element-by-element).

### 4.6 Aliasing in Data Clauses

*Issue 14.* Users have seen different behavior in different implementations when different arrays on data clauses are aliased to the same actual data. The simplest example is `acc parallel loop copyin(a...) copyout(b...)`, where `a` and `b` are pointers or reference arguments that actually point to the same array. Depending on the implementation, the `copyin(a)` may happen while the `copyout(b)` sees that the data is already present, or both `copyin` and `copyout` may happen (which is probably what the user wants and expects). We are discussing how to best resolve this case.

### 4.7 Synonyms for acc loop: acc do and acc for

*Issue 47.* To make porting between OpenMP and OpenACC more straightforward, we have discussed making `acc do` (in Fortran) and `acc for` be synonyms for `acc loop`.

## 4.8 Tasking

*Issue 52.* We are exploring whether and how tasks might make sense in the highly parallel future that OpenACC targets.

## 4.9 Debugger Interface

*Issue 35.* OpenACC added a profiler tool interface in the 2.5 specification. We have also discussed a debugger interface, though we need more input from the debugger community.

© 2016 OpenACC-Standard.org. All rights reserved.