The OpenMP Architecture Review Board

OpenMP Technical Report 1 on Directives for Attached Accelerators

This Technical Report specifies proposed directives for the execution of loops and regions of code on attached accelerators.

Eric Stotzer, Texas Instruments (editor, accelerator subcommittee co-chair): estotzer@ti.com
James Beyer, Cray (accelerator subcommittee co-chair)
Dibyendu Das, AMD
Gabriele Jost, AMD
Prakash Raghavendra, AMD
John Leidel, Convey
Ajejandro Duran, Intel
Ravi Narayanaswamy, Intel
Xinmin Tian, Intel
Oscar Hernandez, Oak Ridge National Labs (ORNL)
Christian Terboven, RWTH Aachen University
Sandra Wienke, RWTH Aachen University
Lars Koesterke, Texas Advanced Computing Center (TACC)
Kent Milfeld, Texas Advanced Computing Center (TACC)
Ajay Jayaraj, Texas Instruments
Robert Dietrich, TU Dresden, ZIH

11/6/2012

Expires November 6, 2015

We actively solicit comments. Please provide feedback on this document either to the Editor directly or in the OpenMP Forum at openmp.org

End of Public Comment Period: Jan 27, 2013

This technical report describes possible future directions or extensions to the OpenMP Specification.

The goal of this technical report is to build more widespread existing practice for an expanded OpenMP. It gives advice on extensions or future directions to those vendors who wish to provide them possibly for trial implementation, allows OpenMP to gather early feedback, support timing and scheduling differences between official OpenMP releases, and offers a preview to users of the future directions of OpenMP wih the provision stated in the next paragraph.

This technical report is non-normative. Some of the components in this technical report may be considered for standardization in a future version of OpenMP, but they are not currently part of any OpenMP Specification. Some of the components in this technical report may never be standardized, others may be standardized in a substantially changed form, or it may be standardized as is in its entirety.

Changes to 1.2.2 OpenMP Language Terminology

Add after [7:33][1]

- **device** A logical execution engine with local storage.

1.2.4 Data Terminology

Change [10:8-10] from:

- **data environment** All the variables associated with the execution of a given task. The data environment for a given task is constructed from the data environment of the generating task at the time the task is generated.

to:

- **data environment** The variables associated with execution of a given region.

- **device data environment** A data environment associated with a **target data** or **target** region.

Change section 1.3 (Execution Model)

Insert new paragraphs following [13:13]:

The thread that executes the initial OpenMP program executes on the *host device*. An implementation may support other *target devices*. If supported, there are one or more devices available to the host device for offloading code and data. Each device has its own threads that are

[1] The **[***page number* **:** *line-range***]** notation refers to pages and line numbers in the OpenMP API v3.1 specification.

distinct from threads executing on another device. Threads cannot migrate from one device to another device. The execution model is host-centric such that the host device offloads **target** regions to target devices.

A **target** region begins as a single thread of execution, called the initial device thread. The initial device thread executes sequentially, as if enclosed in an implicit task region, called the initial device task region that is defined by an implicit inactive **parallel** region surrounding the whole **target** region.

When a **target** construct is encountered, the **target** region is executed by the implicit device task. The task that encounters the **target** construct waits at the end of the construct until execution of the region completes. If a target device does not exist, or the target device is not supported by the implementation, or the target device cannot execute the **target** construct then the **target** region is executed by the host device.

If a construct creates a data environment, it is created at the time the construct is encountered. Whether a construct creates a data environment is defined in the description of the construct.

Change section 1.4.3 OpenMP Memory Consistency

Add after [17:3]:

The original variable in a host task data environment and the corresponding variable(s) in one or more device data environments may share storage. Without intervening synchronization the following cases can result in data races:

- Writes to the corresponding variable by a device thread followed by a read or write of the original variable by a host task.
- Writes to the original variable by a host task followed by a read or write of the corresponding variable by a device thread.
- Writes to the corresponding variable by a thread on one device followed by a read or write of the corresponding variable by a thread on another device.
- Writes to the corresponding variable by a thread on one device followed by a read or write of the corresponding variable by another thread in the same device but in a different **target** region.

Change section 2.7.3 task scheduling

Add after [65:23]:

- immediately after a **target** construct
- immediately after a **target data** construct
- in a **target update** region

Changes to section 2.3.1 ICV Descriptions

Add after [31:24]:

- *device-num-var* - controls the default device. There is one copy of this ICV per data environment.

Change section 2.3.2 Modifying and Retrieving ICV Values

Add lines at end of table Modifying and Retrieving ICV variables [31:1]:

ICV	Scope	Ways to modify values	Way to retrieve value	Initial Value
device-num-var	data environment	**OMP_DEVICE_NUM** **omp_set_device_num()**	**omp_get_device_num()**	Implementation defined

Changes to section 2.3.2

Change [30:14] from:

Clauses on OpenMP constructs do not modify the values of any of the ICVs.

to:

Unless explicitly stated, clauses on OpenMP constructs do not modify the values of any of the ICVs.

Changes to section 2.3.3 How the Per-Data Environment ICVs Work

Change [30:22-23] from:

When a task construct or parallel construct is encountered, the generated task(s) inherit the values of *dyn-var*, *nest-var*, and *run-sched-var* from the generating task's ICV values.

to:

When a task construct, parallel construct or target construct is encountered, the generated task(s) inherit the values of *dyn-var*, *nest-var*, *run-sched-var* and *device-num-var* from the generating task's ICV values. For the target construct, if it has a **device** clause the *device-num-var* is initialized to the value of the clause.

Add new section 2.3.4 How the ICVs work in target regions and renumber sections after

Each device has its own copy of the ICVs with a global scope, and it is implementation defined how they are initialized.

When a **target** construct is encountered, the device data environment inherits the values of *dyn-var*, *nest-var*, *nthreads-var* and *run-sched-var* from the generating task's ICV values.

2.8 target constructs

2.8.1 target data construct

Summary

Create a device data environment for the extent of the region.

Syntax

C/C++
#pragma omp target data *[clause[[,] clause],...] new-line* *structured-block*
C/C++
Fortran
!$omp target data *[clause[[,] clause],...]* *structured-block* **!$omp end target data**
Fortran

Clauses

device(*integer-expression* **)**
map(*list* **)**
mapto(*list* **)**
mapfrom(*list* **)**
scratch(*list* **)**
if(*scalar-expression* **)**

Binding

The binding task region for a **target data** construct is the encountering task. The target region binds to the enclosing parallel or task region.

Description

When a **target data** construct is encountered, a new device data environment is created, and the encountering task executes the **target data** region. If there is no **device** clause, the default device is determined by the *device-num-var* ICV. The new device data environment is constructed from the enclosing data environment and any data motion clauses on the construct. When an **if** clause is present and the **if** clause expression evaluates to *false*, the new device data environment is not created.

Restrictions

- A program must not depend on any ordering of the evaluations of the clauses of the **target data** directive, or on any side effects of the evaluations of the clauses.
- At most one **device** clause may appear on the directive. The **device** expression must evaluate to a positive integer value.
- At most one **if** clause can appear on the directive.

Cross References

- Data Motion clauses, see Section 2.9 on page ##.
- *device-num-var*, see Section 2.3 on page 30.

2.8.2 target construct

Summary

Create a device data environment and execute the construct on the same device.

Syntax

C/C++
#pragma omp target *[clause[[,] clause],...] new-line* *parallel-loop-construct* | *parallel-sections-construct*
C/C++
Fortran
!$omp target** *[clause[[,] clause],...]* *parallel-loop-construct* \| *parallel-sections-construct* **!$omp end target
Fortran

Clauses

device(*integer-expression* **)**
map(*list* **)**
mapto(*list* **)**
mapfrom(*list* **)**

scratch(*list* **)**
num_threads(*list* **)**
if(*scalar-expression* **)**

Binding

The binding task for a **target** construct is the encountering task. The target region binds to the enclosing parallel or task region.

Description

The **target** construct provides a superset of the functionality and restrictions provided by the **target data** construct. In addition, the **target** construct specifies that the region is executed by a device. The encountering task waits for the device to complete the target region at the end of the construct. When an **if** clause is present and the **if** clause expression evaluates to *false*, the target region is not executed by the device, but instead is executed by the encountering task. When a **num_threads** clause is present, the *nthreads-var* in the new device data environment is assigned the value *list*.

Restrictions

- If a **target**, **target update**, or **target data** construct appears within a target region then the construct is ignored.
- The result of an **omp_set_device** or **omp_get_device** routine called within a target region is unspecified.

Cross References

- **target data** construct, see Section 2.8.1 on page ##.
- Data motion clauses, see Section 2.9 on page ##.

2.8.3 target update construct

Summary

The **target update** directive makes the list items in the device data environment consistent with their corresponding original list items.

Syntax

C/C++
#pragma omp target update *[clause[[,] clause],...] new-line*
C/C++
Fortran
!$omp target update *[clause[[,] clause],...]*
Fortran

Clauses

device(*integer-expression* **)**
mapto(*list* **)**
mapfrom(*list* **)**
if(*scalar-expression* **)**

Binding

The binding task for a **target update** construct is the encountering task.

Description

The **target update** directive makes the list items in the device data environment consistent with their corresponding original list items. If the corresponding list item does not exist in the device data environment, the list item is ignored in the **mapto** or **mapfrom**. The device is specified in the **device** clause. If there is no **device** clause, the device is determined by the *device-num-var* ICV.

When an **if** clause is present and the **if** clause expression evaluates to *false*, the **target update** construct is ignored.

Restrictions

- A **target update** shall not appear inside of a **target** region.
- A list item may only appear in a **mapto** or **mapfrom** clause, but not both.
- At most one **device** clause may appear on the directive. The **device** expression must evaluate to a positive integer value.
- At most one **if** clause can appear on the directive.

Cross References

- **mapto** and **mapfrom** clauses, see Section 2.9 on page ##.

2.8.4 declare target directive

Summary

The **declare target** construct can be applied to a function (C, C++ and Fortran) or a subroutine (Fortran) to enable the creation of a device specific version that can be called from a target region.

Syntax

The syntax of the **declare target** construct is as follows:

C/C++
#pragma omp declare target *new-line* *function-definition-or-declaration*
C/C++
Fortran
!$omp declare target *(subroutine-or-function-name)*
Fortran

Description

C/C++
The use of a **declare target** construct on a function enables the creation of a version of the associated function that can be used inside a target region that executes on the device.
C/C++
Fortran
The use of a **declare target** construct enables the creation of a versions of the specified subroutine or function that can be used inside a target region that executes on the device.
Fortran

Restrictions

C/C++
• All declarations and definitions for the function must have a **declare target** construct if one is specified for any of them. Otherwise, the result is unspecified.
C/C++
Fortran
• Procedure pointers may not be used to access versions created by the **declare target** directive.
Fortran

2.8.5 declare target mirror directive

Summary

The **declare target mirror** directive specifies that variables are replicated, with a device having its own copy. The **declare target mirror** directive is a declarative directive.

Syntax

C/C++
#pragma omp declare target mirror(*list* **)** *new-line*
C/C++

Fortran
!$omp declare target mirror(*list* **)**
Fortran

Description

For each list item that appears in a **declare target mirror** directive, a corresponding list item is available in the device data environment for the extent of the program.

If the original list item is initialized, the corresponding list item in the device data environment is initialized with the same value.

Restrictions

2.8.5 declare target linkable directive

Summary

The **declare target linkable** directive specifies that the named variable will be made available on the device. The **declare target linkable** directive is a declarative directive.

Syntax

C/C++
#pragma omp declare target linkable(list) *new-line*
C/C++
Fortran
!$omp declare target linkable(*list* **)**
Fortran

Description

The **declare target linkable** directive specifies that the list item will be made available on the device.

This construct does not move the variable to the device, so the user must use a **target data** or **target** construct to move the variable to the device.

Restrictions

New Section 2.9.5 Data-Motion Attribute Rules

2.9.5.1 Data-Motion Attribute Rules for Variables Referenced in Construct

The data-motion attributes of variables that are referenced in a **target** construct can be *predetermined*, *explicitly determined*, or *implicitly determined*, according to the rules outlined in this section.

Specifying a variable on a clause of an enclosed construct causes an implicit reference to the variable in the enclosing **target** construct. Such implicit references are also subject to the data-motion attribute rules outlined in this section.

Certain variables and objects have *predetermined* data-motion attributes as follows:

C/C++

- Variables appearing in a **declare target mirror** directive have no data-motion attribute.
- Objects with dynamic storage duration have no data-motion attribute.
- Variables with automatic storage duration that are declared in a scope inside the region have no data-motion attribute.
- Variables with static storage duration that are declared in a scope inside the region have no data-motion attribute.

C/C++

Fortran

- Cray pointees inherit the data-motion attribute of the storage with which their Cray pointers are associated.
- An associate name preserves the association with the selector established at the Fortran ASSOCIATE statement.

Fortran

Variables with *predetermined* data-motion attributes may not be listed in data-motion attribute clauses, except for the cases listed below. For these exceptions only, listing a predetermined variable in a data-motion attribute clause is allowed and overrides the variable's predetermined data-sharing attributes.

- Variables appearing in a **declare target mirror** directive can appear in any data-motion clause.

Variables with *explicitly determined* data-motion attributes are those that are referenced in a given construct and are listed in a data-motion attribute clause on the **target** construct.

Variables with *implicitly determined* data-motion attributes are those that are referenced in a given construct, do not have *predetermined* data-motion attributes, and are not listed in a data-motion attribute clause on the construct. All variables that are *implicitly determined* in a **target** construct are map.

2.9.5.2 Data-Motion Attribute Rules for Variables Referenced in Region but not in a Construct

All variables that are referenced in a region, but not in a construct, have no data-motion attribute. A reference to a variable appearing in a **declare target linkable** directive in the region that is not referenced in the construct results in unspecified behavior.

New Section 2.9.6 Data-Motion Clauses

For list items appearing in **map**, **mapto**, **mapfrom**, and **scratch** clauses, corresponding new list items are optionally created in the device data environment associated with the construct.

The original and new list items may share storage such that writes to either item by one task or device followed by a read of the other item by another task or device without intervening synchronization can result in data races.

C/C++
If a new list item is created then a new list item of the same type, with automatic storage duration, is allocated for the construct. The storage and thus lifetime of these list items lasts until the block in which they are created exits. The size and alignment of the new list item are determined by the type of the variable. This allocation occurs, if the region references the list item in any statement. The new list item is initialized, or has an undefined initial value, as if it had been locally declared without an initializer. The order in which any default constructors for new list items of class type are called is unspecified. The order in which any C/C++ destructors for different private variables of class type are called is unspecified
C/C++
Fortran
If a new list item is created then a new list item of the same type, kind, and rank is allocated. The initial value of the new list item is undefined.
Fortran

2.9.6.1 scratch clause

Summary

The **scratch** clause declares one or more list items to be available in the new device data environment.

Syntax

The syntax of the **scratch** clause is as follows:

scratch(*list* **)**

Description

If a corresponding list item of the original list item is in the enclosing device data environment, the new device data environment uses the corresponding list item from the enclosing device data environment.

If a corresponding list item is not in the enclosing device data environment, a new list item with language-specific attributes is derived from the original list item and created in the new device data environment. This new list item becomes the corresponding list item to the original list item in the new device data environment.

Restrictions

- If overlapped variables are used the behavior is unspecified.

2.9.6.2 mapto clause

Summary

The **mapto** clause declares one or more list items to be available in the new device data environment. Each new corresponding list item is initialized with the original list item's value.

Syntax

The syntax of the **mapto** clause is as follows:

mapto(*list* **)**

Description

The **mapto** clause provides a superset of the functionality and restrictions provided by the **scratch** clause.

On entry to the region each new corresponding list item is initialized with the original list item's value.

Restrictions

C/C++
• For variables of non-array type, the map initialization is a bitwise copy.
C/C++
Fortran
• The map initialization is done, as if by assignment.
Fortran

2.9.6.3 mapfrom clause

Summary

The **mapfrom** clause declares one or more list items to be available in the new device data environment, and causes the original list item to be assigned the corresponding list item's value after the end of the region.

Syntax

The syntax of the **mapfrom** clause is as follows:

mapfrom(*list* **)**

Description

The **mapfrom** clause provides a superset of the functionality and restrictions provided by the **scratch** clause.

The original list item is assigned with the corresponding list item's value after the end of the region.

Restrictions

2.9.6.4 map clause

Summary

The **map** clause provides both **mapto** and **mapfrom** functionality for each list item.

Syntax

The syntax of the **map** clause is as follows:

map(*list* **)**

Description

The **map** clause provides a superset of the functionality and restrictions provided by the **mapto** and **mapfrom** clauses.

Restrictions

Changes to section 3.2 Execution Environment Routines

Add to list of routines after p[115:27]:

- the **omp_set_device_num** routine.

- the **omp_get_device_num** routine.

Add after [141:3]

3.2.21 omp_set_device_num

Summary

The **omp_set_device_num** routine assigns the value of the *device-num-var* ICV, which determines the default device number.

Format

C/C++
void omp_set_device_num(int *device_num* **);**
C/C++
Fortran
subroutine omp_set_device_num(*device_num* **)** **integer** *device_num*
Fortran

Binding

The binding task set for an **omp_set_device_num** region is the generating task.

Effect

The effect of this routine is to set the value of the *device-num-var* ICV of the current task to the values specified in the argument.

Cross References:

- *device-num-var*, see Section 2.3 on page 30.
- **omp_get_device_num**, see Section 3.2.22 on page X.
- **OMP_DEVICE_NUM** environment variable, see Section 4.3 on page 168

3.2.22 omp_get_device_num

Summary

The **omp_get_device_num** routine returns the value of the *device-num-var* ICV, which determines the default device number.

Format

C/C++
int omp_get_device_num(void);
C/C++
Fortran
integer function omp_get_device_num()
Fortran

Binding

The binding task set for an **omp_get_device_num** region is the generating task.

Effect

The **omp_get_device_num** routine returns the value of the *device-num-var* ICV of the current task.

Cross References:

- *device-num-var*, see Section 2.3 on page ##.
- **omp_set_device_num**, see Section 3.2.21 on page ##.
- **OMP_DEVICE_NUM** environment variable, see Section 4.3 on page ##.

Changes to Chapter 4

Add after [153:28]

- **OMP_DEVICE_NUM** sets the *device-num-var* ICV that controls the default device number.

Add new section 4.10 OMP_DEVICE_NUM

The **OMP_DEVICE_NUM** environment variable sets the device number to use in target constructs by setting the initial value of the *device-num-var* ICV.

The value of this environment variable must be a positive integer value.

Cross References:

• *device-num-var* ICV, see Section 2.3 on page ##.
• target constructs, Section 2.9