

# GNU Offloading and Multi Processing Runtime Library

---

The GNU OpenMP and OpenACC Implementation

---

Published by the Free Software Foundation  
51 Franklin Street, Fifth Floor  
Boston, MA 02110-1301, USA

Copyright © 2006-2025 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with the Invariant Sections being “Funding Free Software”, the Front-Cover texts being (a) (see below), and with the Back-Cover Texts being (b) (see below). A copy of the license is included in the section entitled “GNU Free Documentation License”.

(a) The FSF’s Front-Cover Text is:

A GNU Manual

(b) The FSF’s Back-Cover Text is:

You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.

## Short Contents

1	Enabling OpenMP.....	1
2	OpenMP Implementation Status .....	3
3	OpenMP Runtime Library Routines .....	15
4	OpenMP Environment Variables .....	59
5	Enabling OpenACC.....	71
6	OpenACC Runtime Library Routines .....	73
7	OpenACC Environment Variables .....	93
8	CUDA Streams Usage.....	95
9	OpenACC Library Interoperability .....	97
10	OpenACC Profiling Interface .....	101
11	OpenMP-Implementation Specifics.....	107
12	Offload-Target Specifics.....	113
13	The libgomp ABI.....	119
14	Reporting Bugs .....	125
	GNU General Public License .....	127
	GNU Free Documentation License.....	139
	Funding Free Software .....	147
	Library Index .....	149



















# 1 Enabling OpenMP

To activate the OpenMP extensions for C/C++ and Fortran, the compile-time flag `-fopenmp` must be specified. For C and C++, this enables the handling of the OpenMP directives using `#pragma omp` and the `[[omp::directive(...)]], [[omp::sequence(...)]]` and `[[omp::decl(...)]]` attributes. For Fortran, it enables for free source form the `!$omp` sentinel for directives and the `!$` conditional compilation sentinel and for fixed source form the `c$omp`, `*$omp` and `!$omp` sentinels for directives and the `c$`, `*$` and `!$` conditional compilation sentinels. The flag also arranges for automatic linking of the OpenMP runtime library (Chapter 3 [Runtime Library Routines], page 15).

The `-fopenmp-simd` flag can be used to enable a subset of OpenMP directives that do not require the linking of either the OpenMP runtime library or the POSIX threads library.

A complete description of all OpenMP directives may be found in the OpenMP Application Program Interface (<https://www.openmp.org>) manuals. See also Chapter 2 [OpenMP Implementation Status], page 3.







































































































































- `$<priority>` is an optional priority for the worker threads of a thread pool according to `pthread_setschedparam`. In case a priority value is omitted, then a worker thread inherits the priority of the OpenMP primary thread that created it. The priority of the worker thread is not changed after creation, even if a new OpenMP primary thread using the worker has a different priority.
- `@<scheduler-name>` is the scheduler instance name according to the RTEMS application configuration.

In case no thread pool configuration is specified for a scheduler instance, then each OpenMP primary thread of this scheduler instance uses its own dynamically allocated thread pool. To limit the worker thread count of the thread pools, each OpenMP primary thread must call `omp_set_num_threads`.

*Example:* Lets suppose we have three scheduler instances `I0`, `WRK0`, and `WRK1` with `GOMP_RTEMS_THREAD_POOLS` set to `"1@WRK0:3$4@WRK1"`. Then there are no thread pool restrictions for scheduler instance `I0`. In the scheduler instance `WRK0` there is one thread pool available. Since no priority is specified for this scheduler instance, the worker thread inherits the priority of the OpenMP primary thread that created it. In the scheduler instance `WRK1` there are three thread pools available and their worker threads run at priority four.



## 5 Enabling OpenACC

To activate the OpenACC extensions for C/C++ and Fortran, the compile-time flag `-fopenacc` must be specified. This enables the OpenACC directive `#pragma acc` in C/C++ and, in Fortran, the `!$acc` sentinel in free source form and the `c$acc`, `*$acc` and `!$acc` sentinels in fixed source form. The flag also arranges for automatic linking of the OpenACC runtime library (Chapter 6 [OpenACC Runtime Library Routines], page 73).

See <https://gcc.gnu.org/wiki/OpenACC> for more information.

A complete description of all OpenACC directives accepted may be found in the OpenACC (<https://www.openacc.org>) Application Programming Interface manual, version 2.6.











































## 7 OpenACC Environment Variables

The variables `ACC_DEVICE_TYPE` and `ACC_DEVICE_NUM` are defined by section 4 of the OpenACC specification in version 2.0. The variable `ACC_PROFLIB` is defined by section 4 of the OpenACC specification in version 2.6.

### 7.1 `ACC_DEVICE_TYPE`

*Description:*

Control the default device type to use when executing compute regions. If unset, the code can be run on any device type, favoring a non-host device type.

Supported values in GCC (if compiled in) are

- `host`
- `nvidia`
- `radeon`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 4.1.

### 7.2 `ACC_DEVICE_NUM`

*Description:*

Control which device, identified by device number, is the default device. The value must be a nonnegative integer less than the number of devices. If unset, device number zero is used.

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 4.2.

### 7.3 `ACC_PROFLIB`

*Description:*

Semicolon-separated list of dynamic libraries that are loaded as profiling libraries. Each library must provide at least the `acc_register_library` routine. Each library file is found as described by the documentation of `dlopen` of your operating system.

*See also:* Section 6.43 [`acc_register_library`], page 91, Chapter 10 [OpenACC Profiling Interface], page 101,

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 4.3.



## 8 CUDA Streams Usage

This applies to the `nvptx` plugin only.

The library provides elements that perform asynchronous movement of data and asynchronous operation of computing constructs. This asynchronous functionality is implemented by making use of CUDA streams<sup>1</sup>.

The primary means by that the asynchronous functionality is accessed is through the use of those OpenACC directives which make use of the `async` and `wait` clauses. When the `async` clause is first used with a directive, it creates a CUDA stream. If an `async-argument` is used with the `async` clause, then the stream is associated with the specified `async-argument`.

Following the creation of an association between a CUDA stream and the `async-argument` of an `async` clause, both the `wait` clause and the `wait` directive can be used. When either the clause or directive is used after stream creation, it creates a rendezvous point whereby execution waits until all operations associated with the `async-argument`, that is, stream, have completed.

Normally, the management of the streams that are created as a result of using the `async` clause, is done without any intervention by the caller. This implies the association between the `async-argument` and the CUDA stream is maintained for the lifetime of the program. However, this association can be changed through the use of the library function `acc_set_cuda_stream`. When the function `acc_set_cuda_stream` is called, the CUDA stream that was originally associated with the `async` clause is destroyed. Caution should be taken when changing the association as subsequent references to the `async-argument` refer to a different CUDA stream.

---

<sup>1</sup> See "Stream Management" in "CUDA Driver API", TRM-06703-001, Version 5.5, for additional information



## 9 OpenACC Library Interoperability

### 9.1 Introduction

The OpenACC library uses the CUDA Driver API, and may interact with programs that use the Runtime library directly, or another library based on the Runtime library, e.g., CUBLAS<sup>1</sup>. This chapter describes the use cases and what changes are required in order to use both the OpenACC library and the CUBLAS and Runtime libraries within a program.

### 9.2 First invocation: NVIDIA CUBLAS library API

In this first use case (see below), a function in the CUBLAS library is called prior to any of the functions in the OpenACC library. More specifically, the function `cublasCreate()`.

When invoked, the function initializes the library and allocates the hardware resources on the host and the device on behalf of the caller. Once the initialization and allocation has completed, a handle is returned to the caller. The OpenACC library also requires initialization and allocation of hardware resources. Since the CUBLAS library has already allocated the hardware resources for the device, all that is left to do is to initialize the OpenACC library and acquire the hardware resources on the host.

Prior to calling the OpenACC function that initializes the library and allocate the host hardware resources, you need to acquire the device number that was allocated during the call to `cublasCreate()`. The invoking of the runtime library function `cudaGetDevice()` accomplishes this. Once acquired, the device number is passed along with the device type as parameters to the OpenACC library function `acc_set_device_num()`.

Once the call to `acc_set_device_num()` has completed, the OpenACC library uses the context that was created during the call to `cublasCreate()`. In other words, both libraries share the same context.

```
/* Create the handle */
s = cublasCreate(&h);
if (s != CUBLAS_STATUS_SUCCESS)
{
    fprintf(stderr, "cublasCreate failed %d\n", s);
    exit(EXIT_FAILURE);
}

/* Get the device number */
e = cudaGetDevice(&dev);
if (e != cudaSuccess)
{
    fprintf(stderr, "cudaGetDevice failed %d\n", e);
    exit(EXIT_FAILURE);
}

/* Initialize OpenACC library and use device 'dev' */
acc_set_device_num(dev, acc_device_nvidia);
```

Use Case 1

---

<sup>1</sup> See section 2.26, "Interactions with the CUDA Driver API" in "CUDA Runtime API", Version 5.5, and section 2.27, "VDPAU Interoperability", in "CUDA Driver API", TRM-06703-001, Version 5.5, for additional information on library interoperability.

### 9.3 First invocation: OpenACC library API

In this second use case (see below), a function in the OpenACC library is called prior to any of the functions in the CUBLAS library. More specifically, the function `acc_set_device_num()`.

In the use case presented here, the function `acc_set_device_num()` is used to both initialize the OpenACC library and allocate the hardware resources on the host and the device. In the call to the function, the call parameters specify which device to use and what device type to use, i.e., `acc_device_nvidia`. It should be noted that this is but one method to initialize the OpenACC library and allocate the appropriate hardware resources. Other methods are available through the use of environment variables and these is discussed in the next section.

Once the call to `acc_set_device_num()` has completed, other OpenACC functions can be called as seen with multiple calls being made to `acc_copyin()`. In addition, calls can be made to functions in the CUBLAS library. In the use case a call to `cublasCreate()` is made subsequent to the calls to `acc_copyin()`. As seen in the previous use case, a call to `cublasCreate()` initializes the CUBLAS library and allocates the hardware resources on the host and the device. However, since the device has already been allocated, `cublasCreate()` only initializes the CUBLAS library and allocates the appropriate hardware resources on the host. The context that was created as part of the OpenACC initialization is shared with the CUBLAS library, similarly to the first use case.

```
dev = 0;

acc_set_device_num(dev, acc_device_nvidia);

/* Copy the first set to the device */
d_X = acc_copyin(&h_X[0], N * sizeof (float));
if (d_X == NULL)
{
    fprintf(stderr, "copyin error h_X\n");
    exit(EXIT_FAILURE);
}

/* Copy the second set to the device */
d_Y = acc_copyin(&h_Y1[0], N * sizeof (float));
if (d_Y == NULL)
{
    fprintf(stderr, "copyin error h_Y1\n");
    exit(EXIT_FAILURE);
}

/* Create the handle */
s = cublasCreate(&h);
if (s != CUBLAS_STATUS_SUCCESS)
{
    fprintf(stderr, "cublasCreate failed %d\n", s);
    exit(EXIT_FAILURE);
}

/* Perform saxpy using CUBLAS library function */
s = cublasSaxpy(h, N, &alpha, d_X, 1, d_Y, 1);
if (s != CUBLAS_STATUS_SUCCESS)
{
```

```
        fprintf(stderr, "cublasSaxpy failed %d\n", s);
        exit(EXIT_FAILURE);
    }

    /* Copy the results from the device */
    acc_memcpy_from_device(&h_Y1[0], d_Y, N * sizeof (float));
```

#### Use Case 2

## 9.4 OpenACC library and environment variables

There are two environment variables associated with the OpenACC library that may be used to control the device type and device number: `ACC_DEVICE_TYPE` and `ACC_DEVICE_NUM`, respectively. These two environment variables can be used as an alternative to calling `acc_set_device_num()`. As seen in the second use case, the device type and device number were specified using `acc_set_device_num()`. If however, the aforementioned environment variables were set, then the call to `acc_set_device_num()` would not be required.

The use of the environment variables is only relevant when an OpenACC function is called prior to a call to `cudaCreate()`. If `cudaCreate()` is called prior to a call to an OpenACC function, then you must call `acc_set_device_num()`<sup>2</sup>

---

<sup>2</sup> More complete information about `ACC_DEVICE_TYPE` and `ACC_DEVICE_NUM` can be found in sections 4.1 and 4.2 of the OpenACC (<https://www.openacc.org>) Application Programming Interface”, Version 2.6.











- `acc_update_device`, `acc_update_device_async`
- `acc_update_self`, `acc_update_self_async`
- `acc_map_data`, `acc_unmap_data`
- `acc_memcpy_to_device`, `acc_memcpy_to_device_async`
- `acc_memcpy_from_device`, `acc_memcpy_from_device_async`











the memory; on Linux, this is in particular the case when the memory placement policy is set to preferred.

- The `access` trait has no effect such that memory is always accessible by all threads. (Except on supported no-host devices.)
- The `sync_hint` trait has no effect.

See also: Chapter 12 [Offload-Target Specifics], page 113,























```

{
    body;
}

```

becomes

```

if (GOMP_single_start ())
    body;
GOMP_barrier ();

```

while

```

#pragma omp single copyprivate(x)
    body;

```

becomes

```

datap = GOMP_single_copy_start ();
if (datap == NULL)
{
    body;
    data.x = x;
    GOMP_single_copy_end (&data);
}
else
    x = datap->x;
GOMP_barrier ();

```

### 13.15 Implementing OpenACC's PARALLEL construct

```

void GOACC_parallel ()

```



## 14 Reporting Bugs

Bugs in the GNU Offloading and Multi Processing Runtime Library should be reported via Bugzilla (<https://gcc.gnu.org/bugzilla/>). Please add "openacc", or "openmp", or both to the keywords field in the bug report, as appropriate.









































## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.





## Library Index

### A

acc\_get\_property ..... 74  
 acc\_get\_property\_string ..... 74

### E

Environment Variable ... 59, 60, 61, 62, 63, 64, 65,  
 66, 67, 68

### F

FDL, GNU Free Documentation License ..... 139

### I

Implementation specific setting.. 63, 65, 66, 68, 107