



NVIDIA DOCA DPACC Compiler

User Guide

Table of Contents

Chapter 1. Introduction.....	1
1.1. Glossary.....	1
1.2. Offloading Work on DPA.....	1
1.3. Writing DPA Applications.....	2
1.3.1. Language Support.....	2
1.3.2. Restrictions on DPA Code.....	2
1.3.3. DPA RPC Functions.....	3
1.3.4. DPA Global Functions.....	3
1.3.5. Characteristics of Annotated Functions.....	3
1.3.6. Handling User-defined Data Types.....	3
1.3.7. Characteristics of Annotated Types.....	3
1.3.8. DPA Intrinsics.....	4
Chapter 2. Prerequisites.....	5
2.1. Supported Versions.....	5
Chapter 3. Description.....	6
3.1. DPACC Inputs and Outputs.....	6
3.1.1. DPA Program.....	7
3.1.2. DPA Object.....	7
3.1.3. DPA Library.....	8
3.2. DPACC Trajectory.....	9
3.3. Modes of Operation.....	10
3.3.1. Compile-and-link Mode.....	10
3.3.2. Compile-only Mode.....	11
3.3.3. Library Generation Mode.....	11
Chapter 4. Execution.....	12
4.1. Mandatory Arguments.....	12
4.2. Commonly Used Arguments.....	13
4.3. LTO Usage Guidelines.....	14
4.3.1. Restrictions.....	14
4.3.2. Compatibility.....	14
4.4. Examples.....	14
4.4.1. Building Libraries.....	15
4.4.2. Linking with DPA Device Library.....	15
4.4.3. Enabling Link-time Optimizations.....	15
4.4.4. Including Headers.....	15

4.5. DPA Compiler Usage.....	15
4.5.1. Compiler Driver Command Line Options.....	16
4.5.2. Linker Command Line Options.....	16
4.5.3. dpacc-extract Command Line Options.....	16
4.5.4. Objdump Command Line Options.....	16
4.5.5. Archiver Command Line Options.....	17
4.5.6. NM Tool Command Line Options.....	17
4.5.7. Common Compiler Options.....	17
4.5.8. Common Linker Options.....	17
4.5.9. Debugging Options.....	17
4.5.10. Miscellaneous Notes.....	18

Chapter 1. Introduction

DPACC is a high-level compiler for the DPA processor which compiles code targeted for the data-path accelerator (DPA) processor into a device executable and generates a DPA program.

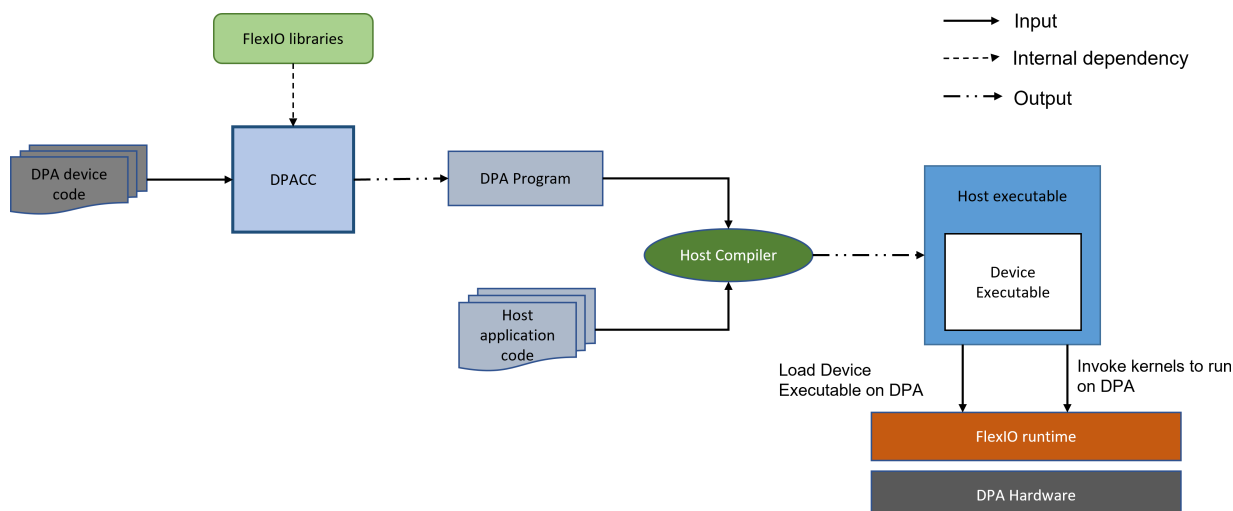
The DPA program is a host library with interfaces encapsulating the device executable. This DPA program is linked with the host application to generate a host executable. The host executable can invoke the DPA code through FlexIO runtime API.

DPACC uses DPA compiler (`dpa-clang`) to compile code targeted for DPA. `dpa-clang` is part of the DPA toolchain package which is an LLVM-based cross-compiling bare-metal toolchain. It provides Clang compiler, LLD linker targeting DPA architecture, and other utilities.

1.1. Glossary

Term	Definition
Device	DPA as present on the BlueField DPU
Host	CPU that launches the device code to run on the DPA
Device function	Any C function that runs on the DPA device
DPA global function	Device function that is the point of entry when offloading any work on DPA
Host compiler	Compiler used to compile the code targeting the host CPU
Device compiler	Compiler used to compile code targeting the DPA
DPA program	Host library that encapsulates the DPA device executable (<code>.elf</code>) and host stubs which are used to access the device executable

1.2. Offloading Work on DPA



To invoke a DPA function from host, the following things are required:

- ▶ DPA device code – C programs, targeted to run on the DPA. DPA device code may contain one or more entry functions.
- ▶ Host application code – the corresponding host application. Please refer to [NVIDIA DOCA DPA Subsystem Programming Guide](#) for more details
- ▶ Runtime – FlexIO or DOCA DPA library provides the runtime

The generated DPA program, when linked with A host application, results in a host executable which also contains the device executable. The host application is in charge of loading the device executable on the device.

1.3. Writing DPA Applications

DPA device code is a C code with some restrictions and special definitions.

FlexIO or DOCA-DPA APIs provide interfaces to DPA.

1.3.1. Language Support

The DPA is programmed using a subset of the C11 language standard. The compiler documents any constructs that are not available. Language constructs, where available, retain their standard definitions.

1.3.2. Restrictions on DPA Code

- ▶ Use of C thread local storage is not allowed for any variables
- ▶ Identifiers with `_dpacc` prefix are reserved by the compiler. Use of such identifiers may result in an error or undefined behavior.
- ▶ DPA processor does not have native floating-point support; use of floating point operations is disabled

1.3.3. DPA RPC Functions

A remote procedure call function is a synchronous call that triggers work in DPA and waits for its completion. These functions return a type `uint64_t` value. They are annotated with a `__dpa_rpc__` attribute.

1.3.4. DPA Global Functions

A DPA global function is an event handler device function referenced from the host code. These functions do not return anything. They are annotated with a `__dpa_global__` attribute.

For more information, refer to the [NVIDIA DOCA DPA Subsystem Programming Guide](#).

1.3.5. Characteristics of Annotated Functions

- ▶ Global functions must have `void` return type and RPC functions must have `uint64_t` return type
- ▶ Annotated functions cannot accept C pointers and arrays as arguments (e.g., `void my_global (int *ptr, int arr[])`)
- ▶ Annotated functions cannot accept a variable number of arguments
- ▶ Inline specifier is not allowed on annotated functions

1.3.6. Handling User-defined Data Types

User-defined data types, when used as global function arguments, require special handling. They must be annotated with a `__dpa_global__` attribute.

If the user-defined data type is `typedef`'d, the `typedef` statement must be annotated with a `__dpa_global__` attribute along the data type itself.

1.3.7. Characteristics of Annotated Types

- ▶ They must have a copy of the definition in all translation units where they are used as global function arguments
- ▶ They cannot have pointers, variable length arrays, and flexible arrays as members
- ▶ Fixed-size arrays as C structure members are supported
- ▶ These characteristics apply recursively to any user-defined/`typedef`'d types that are members of an annotated type

DPACC processes all annotated functions along with annotated types and generates host and device interfaces to facilitate the function launch.

1.3.8. DPA Intrinsic



DPA features such as fences and processor-specific instructions are exposed via intrinsics by the DPA compiler. All intrinsics defined in the header file `dpaintrin.h` are guarded by the `DPA_INTRIN_VERSION_USED` macro. The current `DPA_INTRIN_VERSION` is 1.3.

Example:

```
#define DPA_INTRIN_VERSION_USED (DPA_INTRIN_VERSION(1, 3))
#include <dpaintrin.h>
...
__dpa_thread_writeback_window(); // Fence for write barrier
```

For more information, please refer to [NVIDIA DOCA DPA Subsystem Programming Guide](#).

Chapter 2. Prerequisites

Package	Instructions
Host compiler	<p>Compiler specified through <code>hostcc</code> option. Both <code>gcc</code> and <code>clang</code> are supported.</p> <p> Note: Minimum supported version for <code>clang</code> as <code>hostcc</code> is <code>clang 3.8.0</code>.</p>
Device compiler	<p>The default device compiler is the "DPA compiler". Installing the DPACC package also installs the DPA compiler binaries <code>dpa-clang</code>, <code>dpa-ar</code>, <code>dpa-nm</code> and <code>dpa-objdump</code>.</p> <p> Note: <code>dpa-clang</code> is the only supported device compiler.</p>
FlexIO SDK and C library	<p>Available as part of the DOCA software package. DPA toolchain does not provide C library and corresponding headers. Users are expected to use the C library for DPA from the FlexIO SDK.</p>

2.1. Supported Versions

- ▶ DPACC version 1.5.0
- ▶ See [NVIDIA DOCA DPA Subsystem Programming Guide](#) for other component versions

Chapter 3. Description

3.1. DPACC Inputs and Outputs

DPACC can produce DPA programs in a single command by accepting all source files as input. DPACC also offers the flexibility of producing DPA object files or libraries from input files.

DPA object files contain both host stub objects (DPACC-generated interfaces) and device objects. These DPA object files can later be given to DPACC as input to produce the DPA library.

Phase	Option Name	Default Output File Name
Compile input device code files to DPA object files	<code>--compile</code> or <code>-c</code>	<code>.dpa.o</code> appended to the name of each input source file
Compile and link the input device code files/DPA object files, and produce a DPA program	No specific option	No default name, output file name must be specified
Compile and build DPA library from input device code files/DPA object files	<code>--gen-libs</code> or <code>-gen-libs</code>	No default name, output library name must be specified

DPACC can accept the following file types as input:

File Extension	File Type	Description
<code>.c</code>	C source file	DPA device code
<code>.dpa.o</code>	DPA object file	Object file generated by DPACC, containing both host and device objects
<code>.a</code>	DPA object archive	An archive of DPA object files. User can generate this archive from DPACC-generated DPA objects.

Based on the mode of operations, DPACC can generate the following output files:

Output File Type	Input Files
DPA object file	C source files
DPA program	C source files, DPA object files, and/or DPA object archives
DPA library (DPA host library and DPA device library)	C source files, DPA object files, and/or DPA object archives

The following provides the commands to generate different kinds of supported output file types for each input file type:

Input	Output	DPACC Command
C source file	DPA program	<code>dpacc -hostcc=gcc in.c -o libprog.a</code>
	DPA object	<code>dpacc -hostcc=gcc in.c -c</code>
	DPA library	<code>dpacc -hostcc=gcc in.c -o lib<name> -gen-libs</code>
DPA object	DPA program	<code>dpacc -hostcc=gcc in.dpa.o -o libprog.a</code>
	DPA library	<code>dpacc -hostcc=gcc in.dpa.o -o lib<name> -gen-libs</code>
DPA object archive	DPA program	<code>dpacc -hostcc=gcc in.a -o libprog.a</code>
	DPA library	<code>dpacc -hostcc=gcc in.a -o lib<name> -gen-libs</code>

3.1.1. DPA Program

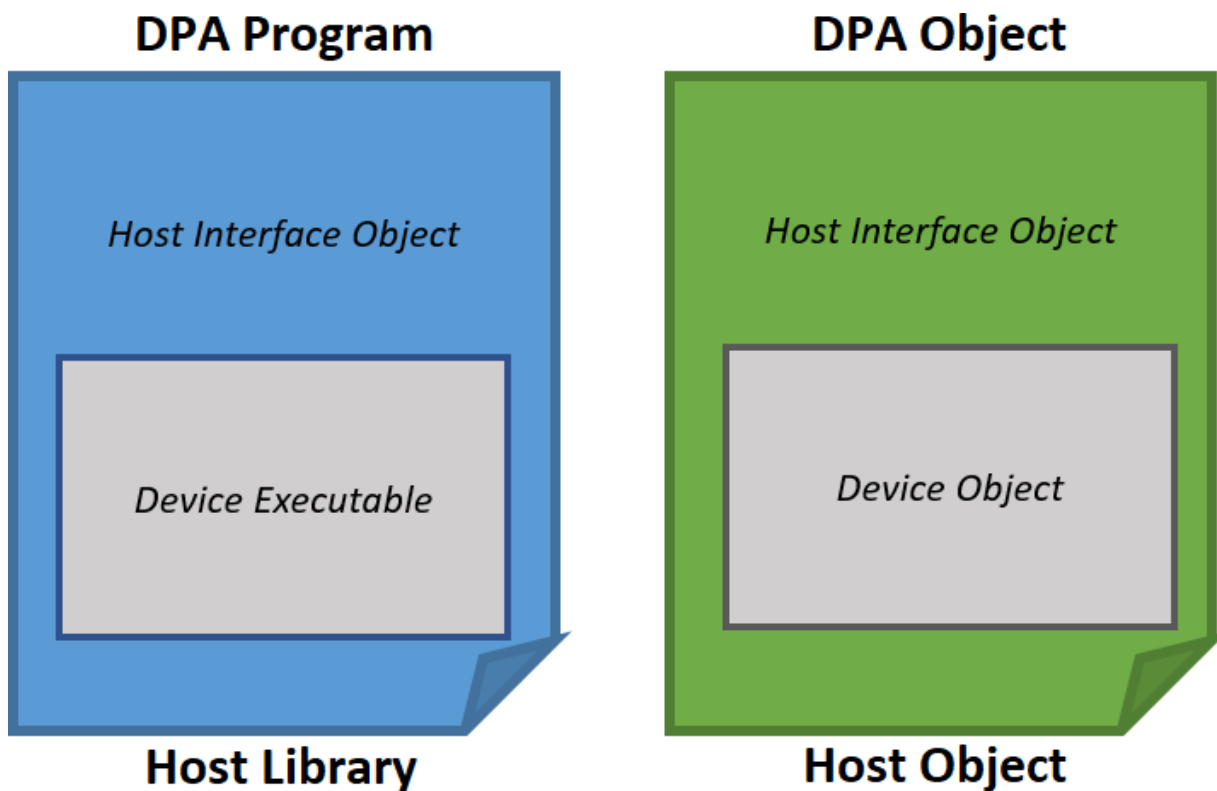
DPACC produces a DPA program in compile-and-link mode. A DPA program is a host library which contains:

- ▶ DPACC-generated host stubs which facilitate invoking a DPA global function from the host application
- ▶ Device executable, generated by DPACC by compiling input DPA device code

DPA program library must be linked with the host application that contains appropriate runtime APIs to load the device executable onto DPA memory.

3.1.2. DPA Object

DPACC produces DPA object files in compile-only mode. A DPA object is an object file for the host machine. In a DPA object, the device object generated by compiling the input device code file is placed inside a specific section of the generated host stubs object. This process is repeated for each input file.



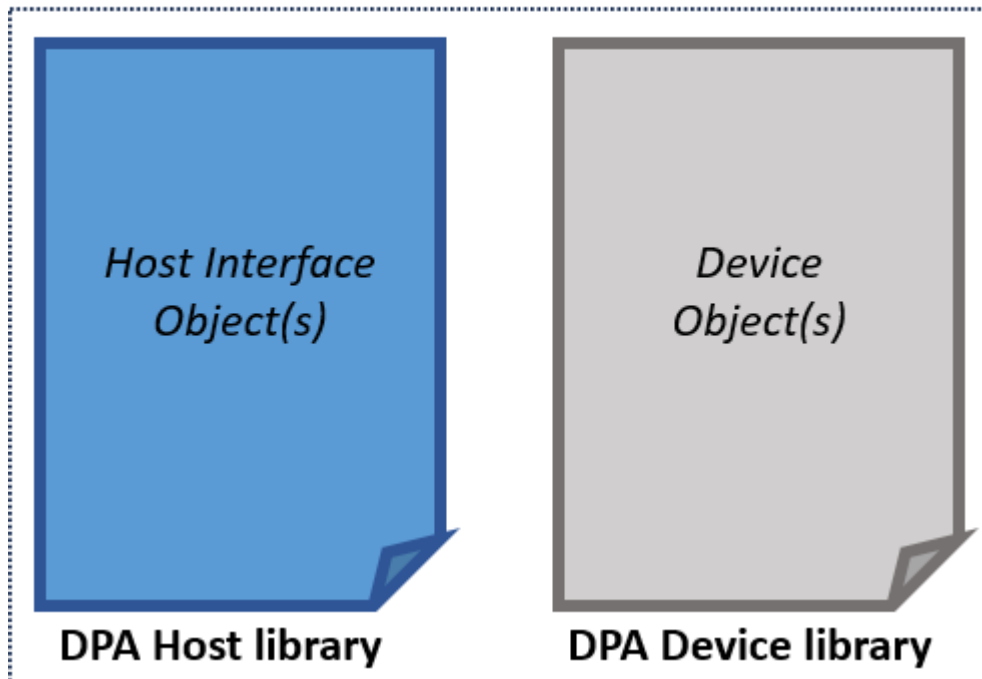
3.1.3. DPA Library

A DPA library is a collection of two individual libraries:

- ▶ DPA device library – contains device objects generated from input files
- ▶ DPA host library – contains host interface objects corresponding to the device objects in DPA device library

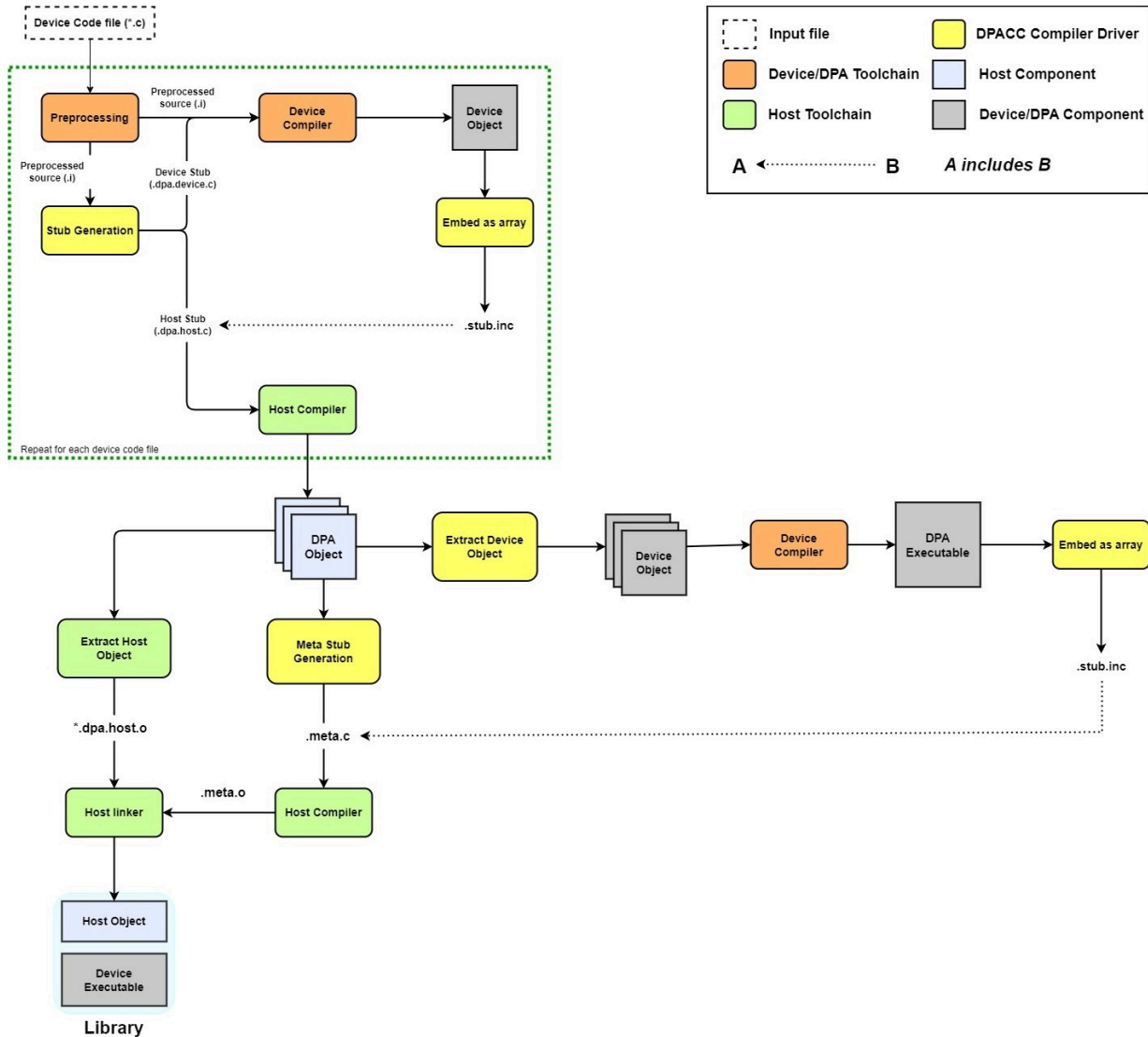
The DPA device library is consumed by DPACC during DPA-program generation and the DPA host library can optionally be linked with other host code and be distributed as the host library. Both libraries are generated as static archives.

DPA Library



3.2. DPACC Trajectory

The following diagram illustrates DPACC compile-and-link mode trajectory.



3.3. Modes of Operation

3.3.1. Compile-and-link Mode

This is a one-step mode that accepts C source files or DPA object files and produces the DPA program. Specifying the output library name is mandatory in this mode.

Example commands:

```
$ dpacc in1.c in2.c -o myLib1.a -hostcc=gcc # Takes C sources to produce myLib1.a library
$ dpacc in3.dpa.o in4.dpa.o -o myLib2.a -hostcc=gcc # Takes DPA object files to produce myLib2.a library
$ dpacc in1.c in3.dpa.o -o myLib3.a -hostcc=gcc # Takes C source and DPA object to produce myLib3.a library
```

3.3.2. Compile-only Mode

This mode accepts C source code and produces `.dpa.o` object files. These files can be given to DPACC to produce the DPA program. The mode is invoked by the `--compile` or `-c` option.

The user can explicitly provide the output object file name using the `--output-file` or `-o` option.

Example commands:

```
$ dpacc -c input1.c -hostcc=gcc # Produces input1.dpa.o
$ dpacc -c input3.c input4.c -hostcc=gcc # Produces input3.dpa.o and
input4.dpa.o
$ dpacc -c input2.c -o myObj.dpa.o -hostcc=gcc # Produces myObj.dpa.o
```

3.3.3. Library Generation Mode

This mode accepts C source files or DPA object files and produces the DPA program. Specifying the output DPA library name is mandatory in this mode.

Example commands:

```
$ dpacc in1.c in2.c -o libdummy1 -hostcc=gcc -gen-libs # Takes C
sources to produce libdummy1_host.a and libdummy_device.a archives
$ dpacc in3.dpa.o in4.dpa.o -o libdummy2 -hostcc=gcc -gen-libs # Takes DPA
object files to produce libdummy2_host.a and libdummy2_device.a archives
$ dpacc in1.c in3.dpa.o -o outdir/libdummy3 -hostcc=gcc -gen-libs # Takes C source
and DPA object to produce outdir/libdummy3_host.a and outdir/libdummy3_device.a
archives
```

Chapter 4. Execution


To execute DOCA DPACC Compiler:

```
Usage: dpacc <list-of-input-files> -hostcc=<path> [other options]
```

Helper Flags:

```
-h, --help           Print help information about DPACC
-V, --version       Print DPACC version information
-v, --verbose       List the compilation commands generated by
                    this invocation while also executing every command in verbose mode
-dryrun, --dryrun   Only list the compilation commands generated
                    by DPACC, without executing them
-keep, --keep       Keep all intermediate files that are generated
                    during internal compilation steps in the current directory
-keep-dir, --keep-dir
                    Keep all intermediate files that are generated
                    during internal compilation steps in the given directory
-optf, --options-file <file>,...
                    Include command line options from the
                    specified file
```


4.1. Mandatory Arguments

Flag	DPACC Mode	Description
List of one or more input files	All	List of C source files or DPA object file names. Specifying at least one input file is mandatory. A file with an unknown extension is treated as a DPA object file.
-hostcc, --hostcc <path>	All	Specify the host compiler. This is typically the native compiler present on the host system.  Note: The host application must be linked with the DPA program using the same compiler or a link-compatible compiler.
-o, --output-file <file>	Compile-and-link/library generation	Specify name and location of the output file.

4.2. Commonly Used Arguments



Tip: Use `--help` option for a list of all supported options.

Flag	Description
<code>-app-name, --app-name <name></code>	Specify DPA application name for the DPA program. This option is required if multiple DPA programs are part of a host application because each DPA application must have a unique name. Default name is <code>__dpa_a_out</code> .
<code>-flto, --flto</code>	Enable link-time optimization (LTO) for device code. Specify this option during compilation along with an optimization level in <code>devicecc-options</code> .
<code>-devicecc-options, --devicecc-options <options>,...</code>	Specify the list of options to pass to the device compiler.
<code>-devicelink-options, --devicelink-options <options>,...</code>	Specify the list of options to pass during device linking stage.
<code>-device-libs, --device-libs '-L<path> -l<name>',...</code>	Specify the list of device libraries including their names (in <code>-l</code>) and their paths (in <code>-L</code>). FlexIO libraries are linked by default.
<code>-I, --common-include-path <path>,...</code>	Specify include search paths common to host and device code compilation. FlexIO headers paths are included by DPACC by default.
<code>-devicecc, --devicecc <path></code>	Specify the device compiler. By default, DPACC invokes dpa-clang .
	<div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;">  WARNING: This option is deprecated and will be removed in the upcoming release. </div>
<code>-o, --output-file <file></code>	Specify name and location of the output file. <ul style="list-style-type: none"> ▶ Compile-only mode – name of the output DPA object file. If not specified, <code>.dpa.o</code> is generated for each <code>.c</code> file. ▶ Compiler-and-link mode – name of the output DPA program. This is a mandatory option in compiler-and-link mode. ▶ Library generation mode – name of the output library. This is a mandatory option for this mode. Output files <code><name>_device.a</code> and <code><name>_host.a</code> are generated.

Flag	Description
<code>-hostcc-options, --hostcc-options <options>, ...</code>	Specify the list of options to pass to the host compiler.
<code>-gen-libs, --gen-libs</code>	Generate a DPA library from input files



Important: The `devicecc-options` option allows passing any option to the device compiler. However, passing options that prevent compilation of the input file may lead to unexpected behavior (e.g., `-devicecc-options="-version"` makes the device compiler print the version and not process input files).



Important: Incompatible options that affect DPA global function argument sizes during DPACC invocation and host application compilation may lead to undefined behavior during execution (e.g., passing `-hostcc-options="-fshort-enums"` to DPACC and missing this option when building the host application).

4.3. LTO Usage Guidelines

4.3.1. Restrictions

- ▶ Only the default linker script is supported with LTO
- ▶ Using options `-fPIC/-fpic/-shared/-mmodel=large` through `-devicecc-options` is not supported when LTO is enabled
- ▶ Fat objects containing both LLVM bitcode and ELF representation are not supported
- ▶ Thin LTO is not supported

4.3.2. Compatibility

During compilation, LLVM generates the object as bitcode IR (intermediate representation) when LTO is enabled instead of ELF representation. The bitcode IR generated by the DPA compiler is only guaranteed to be compatible within the same version of DPACC. All objects involved in link-time optimization (enabled with `-flto`) must be built with the same version of DPACC.

4.4. Examples

This section provides some common use cases of DPACC and showcases the `dpacc` command.

4.4.1. Building Libraries

This example shows how to build DPA libraries using DPACC. Libraries for DPA typically contain two archives, one for the host and one for the device.

```
dpacc input.c -hostcc=gcc -o lib<name> -gen-libs -hostcc-options="-fPIC"
```

This command generates the output files `lib<name>_host.a` and `lib<name>_device.a`.

The host stub archive can be linked with other host code to generate a shared/static host library.

- ▶ Generating a static host library:

```
ar x lib<name>_host.a # Extract objects to generate *.o
ar cr lib<name>.a <*>src.host.o *.o # Generate final static archive with all
objects
```

- ▶ Generating a shared host library:

```
gcc -shared -o lib<name>.so <*>src.host.o -Wl,-whole-archive -l<name>_host -Wl,-
no-whole-archive # Link the generated archive to build a shared library
```

4.4.2. Linking with DPA Device Library

The DPA device library generated by DPACC using `-gen-libs` as part of a DPA library can be consumed by DPACC using the `-device-libs` option.

```
dpacc input.c -hostcc=gcc -o libInput.a -device-libs="-L <path-to-library> -
l<libName>"
```

4.4.3. Enabling Link-time Optimizations

Link-time optimizations can be enabled using `-flto` along with an optimization level specified for device compilation.

```
dpacc input1.c -hostcc=gcc -c -flto -devicecc-options="-O2"
dpacc input2.c -hostcc=gcc -c -flto -devicecc-options="-O2"
dpacc input1.dpa.o input2.dpa.o -hostcc=gcc -o libInput.a
```

4.4.4. Including Headers

This example includes headers for device compilation using `devicecc-options` and host compilation using `hostcc-options`. You may also specify headers for any compilation on both the host and device side using the `-I` option.

```
dpacc input.c -hostcc=gcc -o libInput.a -I <common-headers-path> -devicecc-
options="-I <device-headers-path>" -hostcc-options="-I <host-headers-path>"
```

4.5. DPA Compiler Usage

`dpa-clang` is a compiler driver for accessing the Clang/LLVM compiler, assembler, and linker which accepts C code files or object files and generates an output according to different usage modes.



Note: Invoking the compiler, assembler, or linker directly may lead to unexpected errors.

Refer to the following resources for detailed user guide and command line references:

- ▶ [Clang Compiler User's Manual](#)
- ▶ [Clang command line argument reference](#)
- ▶ [Target-dependent compilation options](#)

4.5.1. Compiler Driver Command Line Options

```
dpa-clang <list-of-input-files> [other-options]
```

4.5.2. Linker Command Line Options

LLD is the default linker provided in the DPA toolchain. Linker-related options are passed to through the compiler driver.

```
dpa-clang -Wl,<linker-option>
```

For more information, please refer to the [LLD command line reference](#).

4.5.3. dpacc-extract Command Line Options

dpacc-extract is a tool for extracting a device executable out of a DPA program or a host executable containing DPA program(s).

To execute dpacc-extract:

```
Usage: dpacc-extract <input-file> -o=<output-file> [other options]
```

Helper Flags:

-o, --output-file	Specify name of the output file
-app-name, --app-name <name>	Specify name of the DPA application to extract
-h, --help	Print help information about dpacc-extract
-V, --version	Print dpacc-extract version information
-optf, --options-file <file>,...	Include command line options from the specified file

Mandatory arguments:

Flag	Description
Input file	DPA program or host executable containing DPA program. Specifying one input file is mandatory.
-o, --output-file <file>	Specify name and location of the output device executable.
-app-name, --app-name <name>	Specify name of the DPA application to extract. Mandatory if input file has multiple DPA apps.

4.5.4. Objdump Command Line Options

The dpa-objdump utility prints the contents of object files and final linked images named on the command line.

For more information, please refer to the [Objdump command line reference](#).

Commonly used dpa-objdump options:

Flag	Description
<code>--mcpu=nv-dpa-bf3</code>	Option to choose micro-architecture for DPA processor. <code>nv-dpa-bf3</code> is the default CPU for <code>dpa-objdump</code> .

4.5.5. Archiver Command Line Options

`dpa-ar` is a Unix `ar`-compatible archiver.

For more information, please refer to the [Archiver command line reference](#).

4.5.6. NM Tool Command Line Options

The `dpa-nm` utility lists the names of symbols from object files and archives.

For more information, please refer to the [NM tool command line reference](#).

4.5.7. Common Compiler Options

Flag	Description
<code>-mcpu=nv-dpa-bf3</code>	Option to choose micro-architecture and ABI for DPA processor. <code>nv-dpa-bf3</code> is the default CPU for the compiler.
<code>-mrelax/-mno-relax</code>	Option to enable/disable linker relaxations.
<code>-I <dir></code>	Option to include header files present in <code><dir></code> .

4.5.8. Common Linker Options

Flag	Description
<code>-Wl, -L <path-to-library> -Wl, -l<library-name></code>	Option to link against libraries



Note: Linker options are provided through the compiler driver `dpa-clang`.



Note: The LLD linker script is honored in addition to the default configuration rather than replacing the whole configuration like in GNU ID. Hence, additional options may be required to override some default behaviors.

4.5.9. Debugging Options

Flag	Description
<code>-fdebug-macro</code>	Option to emit macro debugging information. This option enables macro-debugging similar to GCC option <code>-g3</code> .

4.5.10. Miscellaneous Notes

- ▶ Objects produced by LLD are not compatible with those generated by any other linker.
- ▶ The default debugging standard of the DPA compiler is DWARFv5. GDB versions <10.1 have issues processing some DWARFv5 features. Use the option `-devicecc-options="-gdwarf-4"` with DPACC to debug with GDB versions <10.1.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2023 NVIDIA Corporation & affiliates. All rights reserved.