IBM

Getting Started with CUDA Fortran programming using XL Fortran for Little Endian Distributions

Version 15.1.5

IBM XL Fortran for Linux, V15.1.5

Getting Started with CUDA Fortran programming using XL Fortran for Little Endian Distributions

Version 15.1.5

Contents

About this document

This document is intended for Fortran developers who use XL Fortran to develop CUDA Fortran programs to exploit NVIDIA GPU on OpenPOWER systems.

This document contains detailed information about the CUDA Fortran support that is provided in XL Fortran, including the compiler flow for CUDA Fortran programs, compilation commands, useful compiler options and macros, supported CUDA Fortran features, and limitations.

For general information about XL Fortran, refer to the resources that are listed in "IBM XL Fortran information" on page ix.

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM® XL Fortran for Linux, V15.1.5 information.

Table 1. Typographical conventions

Typeface	Indicates	Example
lowercase bold	Invocation commands, executable names, and compiler options.	The compiler provides basic invocation commands, **xlf**, along with several other compiler invocation commands to support various Fortran language levels and compilation environments. The default file name for the executable program is **a.out**.
italics	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	Make sure that you update the *size* parameter if you return more than the *size* requested.
<u>underlining</u>	The default setting of a parameter of a compiler option or directive.	nomaf | <u>maf</u>
`monospace`	Examples of program code, reference to program code, file names, path names, command strings, or user-defined names.	To compile and optimize myprogram.f, enter: `xlf myprogram.f -03`.
UPPERCASE bold	Fortran programming keywords, statements, directives, and intrinsic procedures. Uppercase letters may also be used to indicate the minimum number of characters required to invoke a compiler option/suboption.	The **ASSERT** directive applies only to the **DO** loop immediately following the directive, and not to any nested **DO** loops.

Qualifying elements (icons and bracket separators)

In descriptions of programming models, this information uses icons to delineate segments of text as follows:

Table 2. Qualifying elements

Icon	Meaning
▶CUDA Fortran CUDA Fortran◀	The text describes CUDA Fortran, the CUDA Fortran support provided by IBM XL Fortran, or both.
▶ GPU GPU ◀	The text describes the information that is relevant to offloading computations to the NVIDIA GPUs.

Syntax diagrams

Throughout this information, diagrams illustrate XL Fortran syntax. This section helps you to interpret and use those diagrams.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

   The ►►── symbol indicates the beginning of a command, directive, or statement.

   The ──→ symbol indicates that the command, directive, or statement syntax is continued on the next line.

   The ►── symbol indicates that a command, directive, or statement is continued from the previous line.

   The ──►◄ symbol indicates the end of a command, directive, or statement.

   Fragments, which are diagrams of syntactical units other than complete commands, directives, or statements, start with the │── symbol and end with the ──│ symbol.
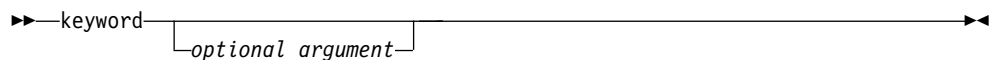
   IBM XL Fortran extensions are marked by a number in the syntax diagram with an explanatory note immediately following the diagram.

   Program units, procedures, constructs, interface blocks and derived-type definitions consist of several individual statements. For such items, a box encloses the syntax representation, and individual syntax diagrams show the required order for the equivalent Fortran statements.

- Required items are shown on the horizontal line (the main path):

   ►►──keyword──*required_argument*─────────────────────────────────►◄

- Optional items are shown below the main path:

   ►►──keyword──►◄
   　　　　　　└─*optional_argument*─┘

   Note: Optional items (not in syntax diagrams) are enclosed by square brackets ([and]). For example, [UNIT=]u

- If you can choose from two or more items, they are shown vertically, in a stack.

   If you *must* choose one of the items, one item of the stack is shown on the main path.

```
►►──keyword──┬─required_argument1─┬────────────────────────────►◄
             └─required_argument2─┘
```
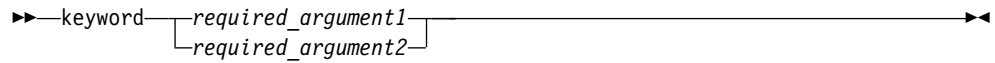
If choosing one of the items is optional, the entire stack is shown below the main path.

```
►►──keyword────────────────────────────────────────────────────►◄
             ┌─optional_argument1─┐
             └─optional_argument2─┘
```

- An arrow returning to the left above the main line (a repeat arrow) indicates that you can make more than one choice from the stacked items or repeat an item. The separator character, if it is other than a blank, is also indicated:

```
             ┌─,────────────────┐
►►──keyword──┴─repeatable_argument─────────────────────────────►◄
```

- The item that is the default is shown above the main path.

```
             ┌─default_argument───┐
►►──keyword──┴─alternate_argument──┴───────────────────────────►◄
```

- Keywords are shown in nonitalic letters and should be entered exactly as shown.
- Variables are shown in italicized lowercase letters. They represent user-supplied names or values. If a variable or user-specified name ends in _list_, you can provide a list of these terms separated by commas.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sample syntax diagram

The following is an example of a syntax diagram with an interpretation:

```
              (1)
►►──EXAMPLE────char_constant──┬─a─┬──┬───┬──────▼─e─┬──name_list───────►◄
                              └─b─┘  ├─c─┤           │
                                     └─d─┘           └──────,──────┘
```

Notes:

1   IBM extension

Interpret the diagram as follows:
- Enter the keyword EXAMPLE.
- EXAMPLE is an IBM extension.
- Enter a value for *char_constant*.
- Enter a value for *a* or *b*, but not for both.
- Optionally, enter a value for *c* or *d*.
- Enter at least one value for *e*. If you enter more than one value, you must put a comma between each.
- Enter the value of at least one *name* for *name_list*. If you enter more than one value, you must put a comma between each. (The *_list* syntax is equivalent to the previous syntax for *e*.)

How to read syntax statements

Syntax statements are read from left to right:
- Individual required arguments are shown with no special notation.
- When you must make a choice between a set of alternatives, they are enclosed by { and } symbols.
- Optional arguments are enclosed by [and] symbols.
- When you can select from a group of choices, they are separated by | characters.
- Arguments that you can repeat are followed by ellipses (...).

Example of a syntax statement

EXAMPLE *char_constant* {a|b}[c|d]e[,e]... *name_list*{name_list}...

The following list explains the syntax statement:
- Enter the keyword EXAMPLE.
- Enter a value for *char_constant*.
- Enter a value for *a* or *b*, but not for both.
- Optionally, enter a value for *c* or *d*.
- Enter at least one value for *e*. If you enter more than one value, you must put a comma between each.
- Optionally, enter the value of at least one *name* for *name_list*. If you enter more than one value, you must put a comma between each *name*.

Note: The same example is used in both the syntax-statement and syntax-diagram representations.

IBM XL Fortran information

XL Fortran provides product information in the following formats:

- Quick Start Guide

  The Quick Start Guide (`quickstart.pdf`) is intended to get you started with IBM XL Fortran for Linux, V15.1.5. It is located by default in the XL Fortran directory and in the `\quickstart` directory of the installation DVD.

- README files

  README files contain late-breaking information, including changes and corrections to the product information. README files are located by default in the XL Fortran directory, and in the root directory and subdirectories of the installation DVD.

- Installable man pages

  Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL Fortran for Linux, V15.1.5 Installation Guide*.

- Online product documentation

  The fully searchable HTML-based documentation is viewable in IBM Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.5/com.ibm.compilers.linux.doc/welcome.html.

- PDF documents

  PDF documents are available on the web at http://www.ibm.com/support/docview.wss?uid=swg27036672.

  The following files comprise the full set of XL Fortran product information:

Table 3. XL Fortran PDF files

Document title	PDF file name	Description
IBM XL Fortran for Linux, V15.1.5 Installation Guide, GC27-6580-04	`install.pdf`	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
Getting Started with IBM XL Fortran for Linux, V15.1.5, SC27-6620-04	`getstart.pdf`	Contains an introduction to the XL Fortran product, with information about setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
IBM XL Fortran for Linux, V15.1.5 Compiler Reference, SC27-6610-04	`compiler.pdf`	Contains information about the various compiler options and environment variables.
IBM XL Fortran for Linux, V15.1.5 Language Reference, SC27-6590-04	`langref.pdf`	Contains information about the Fortran programming language as supported by IBM, including language extensions for portability and conformance to nonproprietary standards, compiler directives and intrinsic procedures.

Table 3. XL Fortran PDF files (continued)

Document title	PDF file name	Description
IBM XL Fortran for Linux, V15.1.5 Optimization and Programming Guide, SC27-6600-04	proguide.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls, floating-point operations, input/output, application optimization and parallelization, and the XL Fortran high-performance libraries.
Getting Started with CUDA Fortran programming using IBM XL Fortran for Linux, V15.1.5, GI13-3562-01	getstart_cudaf.pdf	Contains detailed information about the CUDA Fortran support that is provided in XL Fortran, including the compiler flow for CUDA Fortran programs, compilation commands, useful compiler options and macros, supported CUDA Fortran features, and limitations.

To read a PDF file, use Adobe Reader. If you do not have Adobe Reader, you can download it (subject to license terms) from the Adobe website at http://www.adobe.com.

More information related to XL Fortran, including IBM Redbooks® publications, white papers, and other articles, is available on the web at http://www.ibm.com/support/docview.wss?uid=swg27036672.

For more information about the compiler, see the XL compiler on Power® community at http://ibm.biz/xl-power-compilers.

Technical support

Additional technical support is available from the XL Fortran Support page at http://www.ibm.com/support/entry/portal/product/rational/xl_fortran_for_linux. This page provides a portal with search capabilities to a large selection of Technotes and other support information.

If you cannot find what you need, you can send an email to compinfo@ca.ibm.com.

For the latest information about XL Fortran, visit the product information site at http://ibm.biz/xlfortran-linux.

How to send your comments

Your feedback is important in helping us to provide accurate and high-quality information. If you have any comments about this information or any other XL Fortran information, send your comments to compinfo@ca.ibm.com.

Be sure to include the name of the manual, the part number of the manual, the version of XL Fortran, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

Chapter 1. Overview of CUDA Fortran supported by XL Fortran

CUDA is a parallel programming model and software environment to exploit the NVIDIA GPUs. It provides programmers with a set of instructions that enables GPU acceleration for data-parallel computations. You can increase computing performance of many applications by using CUDA directly or by linking to GPU-accelerated libraries.

IBM XL Fortran for Linux, V15.1.5 supports the CUDA Fortran programming model, which is a subset of the CUDA constructs to exploit the NVIDIA GPUs. You can use the commonly used subset of CUDA Fortran that is provided by IBM XL Fortran for Linux, V15.1.5 to offload computations to the NVIDIA GPUs.

Chapter 2. System prerequisites to compile CUDA Fortran programs

To compile CUDA Fortran programs with IBM XL Fortran for Linux, V15.1.5, you must ensure that your hardware, operating system, and software meet these requirements.

Hardware requirements

You can use any IBM Power Systems™ server that has one or more NVIDIA GPUs installed and is supported by your Linux operating system distribution and NVIDIA CUDA Toolkit. For example, you can use IBM POWER® System S822LC for high performance computing or IBM POWER System S824L. For a complete list of the IBM Power Systems servers, see http://www.ibm.com/systems/power/hardware/.

Supported platforms

You can use the following little endian operating systems supported by the IBM Power Systems servers and NVIDIA CUDA Toolkit 8.0:
- Red Hat Enterprise Linux 7.3 (RHEL 7.3)
- Ubuntu 16.04.1

Software requirements
- NVIDIA CUDA Toolkit 8.0, which you can download from https://developer.nvidia.com/cuda-downloads

  Note: To install the CUDA Toolkit, use the Package Manager installation. The Runfile installation is currently not supported on Power processors. For instructions about Package Manager installation, see the NVIDIA CUDA Installation Guide for Linux (http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html).

Chapter 3. Compiler flow

You can customize the compilation and linking process after understanding the compiler flow. For CUDA Fortran programs, XL Fortran splits host and device code at the intermediate language level rather than the source level.

The compilation process

When CUDA Fortran support is enabled, the compilation process is as follows:

1. Source files, which might contain host code, device code, or both, are processed into the XL intermediate language (W-Code) by the Fortran Frontend (xlfentry) and Transformer (xlfhot).
2. If -O3, -qhot, or -qsmp=omp is in effect, the W-Code is optimized by the High-Level Optimizer (ipa).
3. The W-Code is split into a host W-Code stream and a device W-Code stream by the W-Code intermediate language splitter (partitioner).
   - Host code processing

     The host W-Code stream is processed into an object file by the Low-Level Optimizer (xlfcode).
   - Device code processing
     a. The device W-Code stream is converted to NVVM intermediate representation (NVVM-IR) by the W-Code to LLVM-IR Translator (wc2llvm).
     b. The NVVM-IR is converted to PTX (NVIDIA's low-level Parallel Thread eXecution) instructions by the NVVM-IR to PTX translator (llvm2ptx).
     c. The PTX instructions are assembled into a device object (cubin) by the PTX assembler (ptxas).
     d. The device object is embedded into an object file by the fatbinary tool from the CUDA Toolkit.
4. The object files for the host and device code are combined into a single object file by the host linker (ld).

The following diagram shows the compilation process for CUDA Fortran programs.
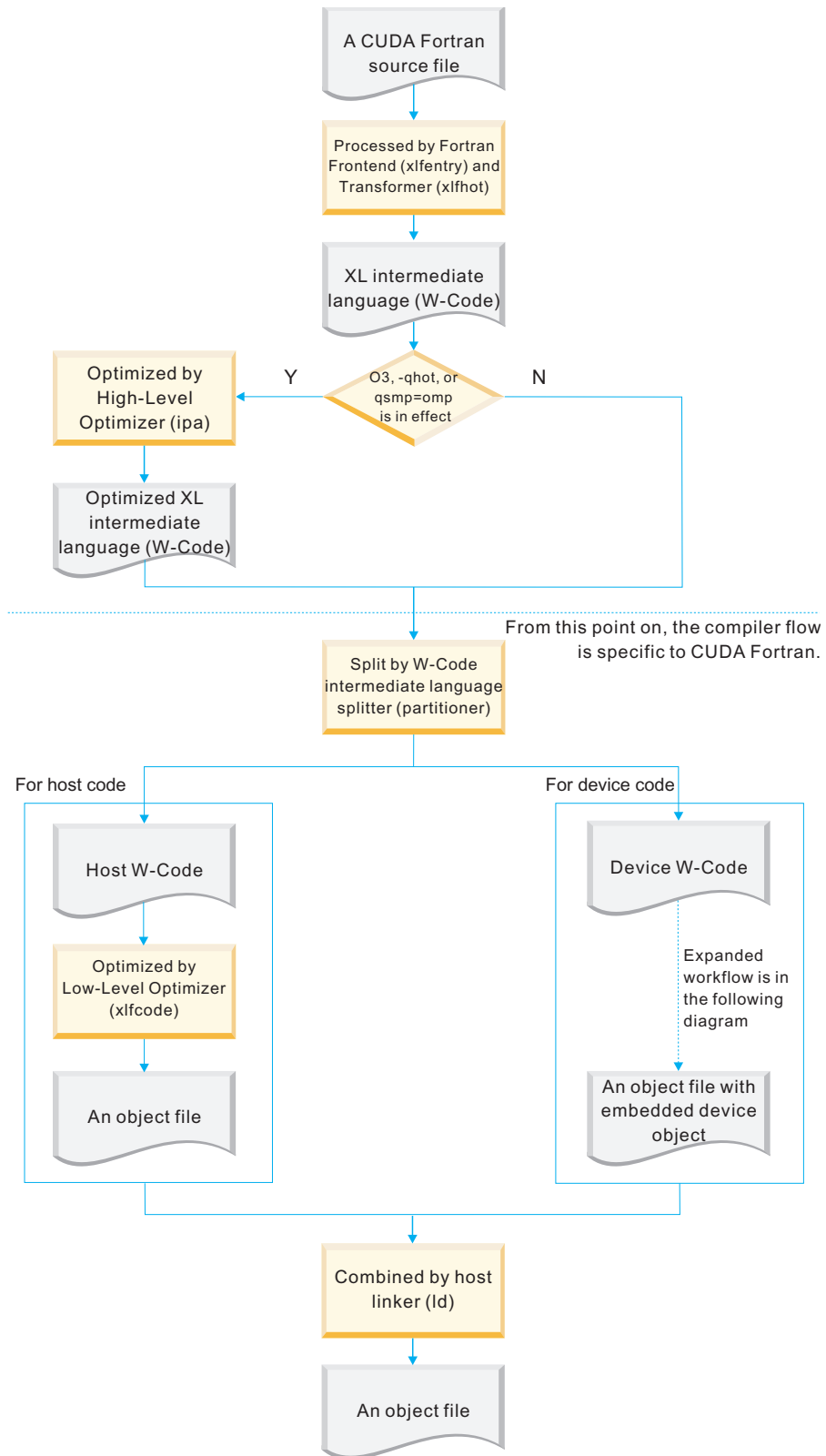
Figure 1. Compilation process for CUDA Fortran programs

The following diagram describes the compilation process for device W-code in detail.

Device W-Code

Converted by W-Code
to LLVM-IR Translator
(wc2llvm)

NVVM intermediate
representation
(NVVM-IR)

Coverted by NVVM-IR
to PTX translator
(llvm2ptx)

PTX instructions

Assembled by PTX
assembler (ptxas)

A device object file
(cubin)

Processed by
fatbinary tool

An object file with
embedded device
object
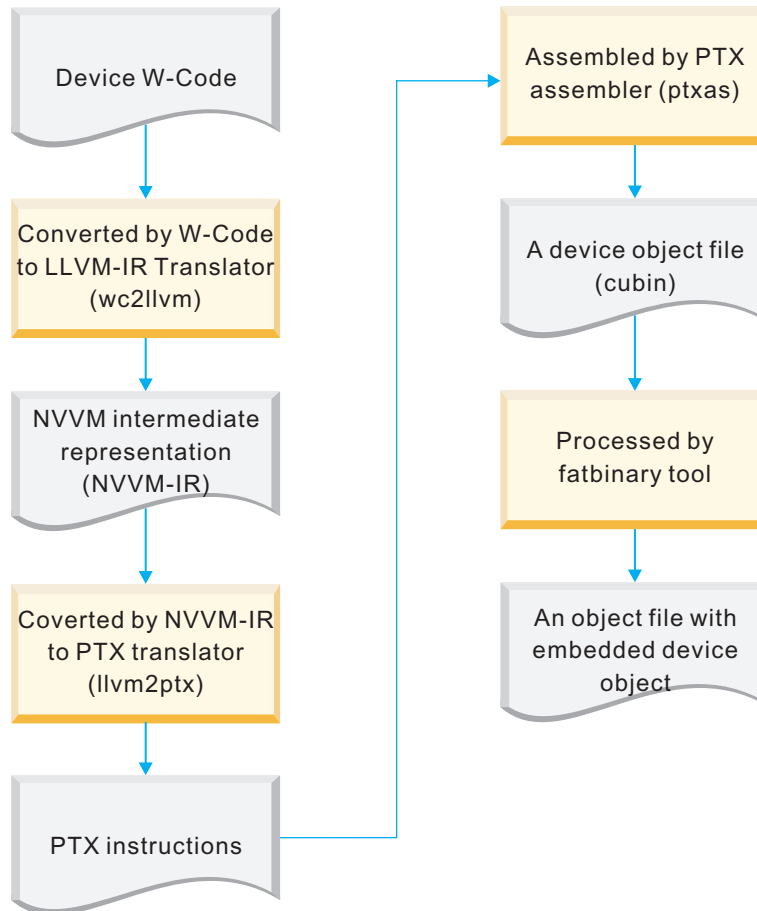
Figure 2. Compilation process for device W-code

The linking process

When CUDA Fortran support is enabled, the linking process is as follows:

1. Embedded device code in the object files is linked into an executable device object by the device linker (nvcc -dlink) from the CUDA Toolkit.
2. The object files and the executable device object are linked into an executable ELF program by the host linker.
3. XL Fortran libraries for the host and the device and several libraries from the CUDA Toolkit are automatically passed to the device and host linkers.

Libraries specified using the **-L** or **-l** option are passed to both the device linker and the host linker.

The following diagram shows the linking process for CUDA Fortran programs.
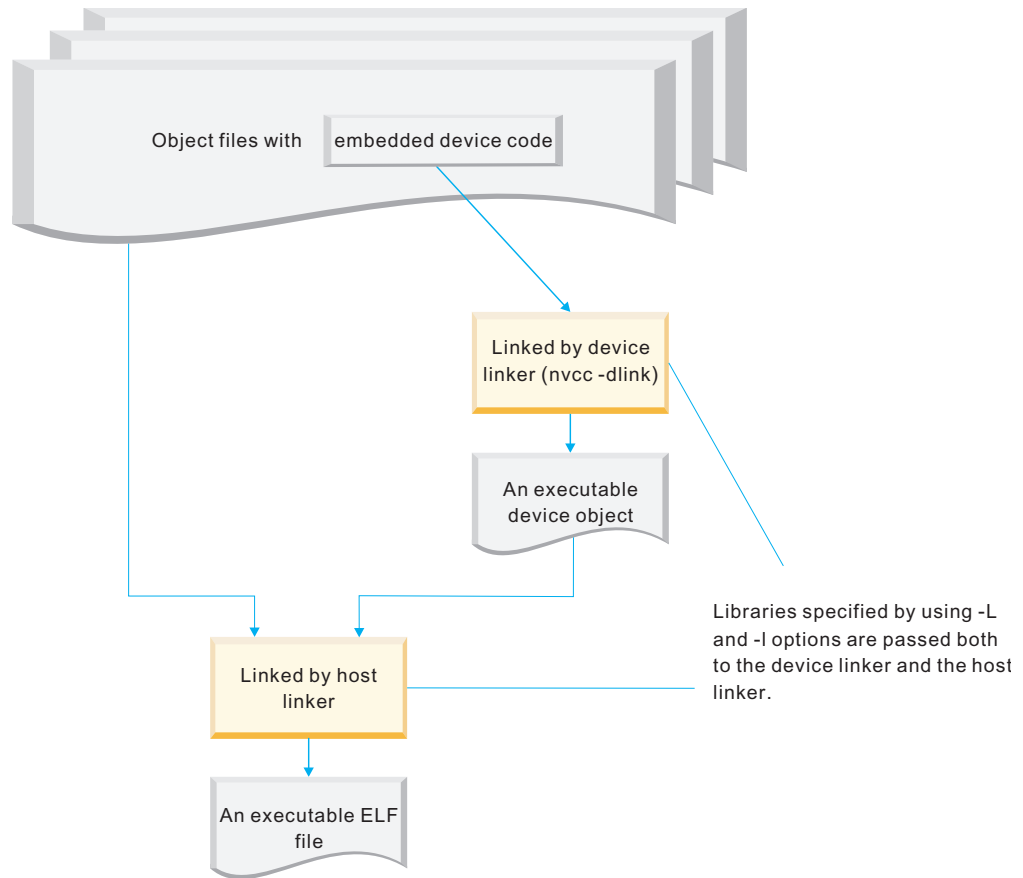
Figure 3. Linking process for CUDA Fortran programs

Abbreviation reference

The following abbreviations and full names are provided for your quick retrieval.

Table 4. Abbreviation of compiler components

Abbreviation	Component
ipa	The High-Level Optimizer
ld	The host linker
llvm2ptx	The NVVM-IR to PTX translator
partitioner	The W-Code intermediate language splitter
ptxas	The PTX assembler
nvcc -dlink	The device linker
wc2llvm	The W-Code to LLVM-IR Translator
xlfentry	The Fortran Frontend
xlfhot	The Fortran Transformer
xlfcode	The Low-Level Optimizer

Table 5. Abbreviation of output and input

Abbreviation	Output/input
cubin	The device object

Table 5. Abbreviation of output and input (continued)

Abbreviation	Output/input
NVVM-IR	The NVVM intermediate representation
PTX	Parallel Thread eXecution
W-Code	The XL intermediate language

Chapter 4. Compiling CUDA Fortran programs

You can use one of the following methods to enable CUDA Fortran support:

- Use the **xlcuf** invocation command to compile any Fortran source files. Specifying **xlcuf** is equivalent to specifying **xlf2008_r** and the **-qcuda** option.
- Use the **xlf_r** invocation command to compile source files with the `.cuf` or `.CUF` suffixes. The **xlf_r** invocation command recognizes `.cuf` and `.CUF` files as CUDA Fortran files and enables CUDA Fortran support automatically.
- Use any threadsafe invocation command, such as **xlf95_r**, and specify the **-qcuda** option to compile any Fortran source files.

Files with the `.cuf` or `.CUF` file extensions are recognized as Fortran source files. The preprocessor is called on `.CUF` files before compilation.

Examples

You can use any of the following commands to compile a CUDA Fortran program:

```
xlcuf myprogram_1.f

xlf_r myprogram_2.cuf

xlf2008_r -qcuda myprogram_3.f
```

Passing options to the CUDA Toolkit

You can change the optimization level of device code or control the strictness of floating-point computation by passing options to the CUDA Toolkit components that are invoked by the compiler.

Passing NVVM options

```
                      (1)
►►──┬──-Xllvm2ptx────────┬──-nvvm-compile-options──=──NVVM_option────────────────►◄
    └──-Wx──,────────────┘
```
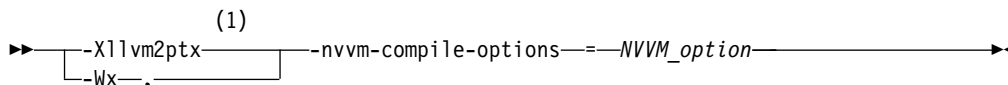
Notes:

1   You must insert one space after **-Xllvm2ptx**.

You can find the NVVM-IR to PTX translator options in the libNVVM API section in the CUDA Toolkit documentation at http://docs.nvidia.com/cuda/libnvvm-api/group__compilation.html under nvvmCompileProgram. The most commonly used options are as follows:

- **-opt**[1]
- **-ftz**
- **-prec-sqrt**
- **-prec-div**
- **-fma**

Note: 1. The default optimization level used by the NVVM backend is **-opt=3**.

To enable fast math on the device, set **ftz** to 1, **prec-sqrt** to 0, **prec-div** to 0, and **fma** to 1 with either of the following equivalent option settings:

- `-Wx,-nvvm-compile-options=-ftz=1,-nvvm-compile-options=-prec-sqrt=0,\`
  `-nvvm-compile-options=-prec-div=0,-nvvm-compile-options=-fma=1`
- `-qxflag=device_fast_math`

Passing ptxas options

```
                    (1)
►►──┬──-Xptxas──────┬──ptxas_option──────────────────────────────────────►◄
    └──-W@──,───────┘
```

Notes:

1   You must insert one space after **-Xptxas**.

The default optimization level used by the PTX assembler is **-O3**.

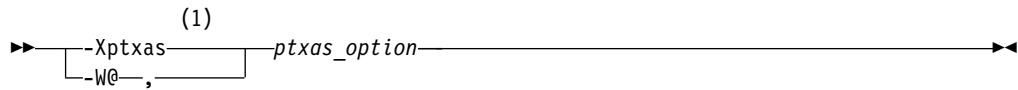You can get a list of the PTX assembler options by running **ptxas** from the CUDA Toolkit with **-h**.

Examples

To disable optimization of device code, you can pass options to the NVVM-IR to PTX translator and the PTX assembler by using either of the following commands:

```
xlcuf mycudaprogram.cuf -Xllvm2ptx -nvvm-compile-options=opt=0 -Xptxas -O0
```

```
xlcuf mycudaprogram.cuf -Wx,-nvvm-compile-options=opt=0 -W@,-O0
```

where **-nvvm-compile-options=opt=0** is a NVVM-IR to PTX translator option, and **-O0** is a PTX assembler option.

Linking to libraries in the CUDA Toolkit

When CUDA Fortran support is enabled, the compiler automatically links in the following libraries from the CUDA Toolkit:

- `libcudadevrt.a`
- `libcudart.so`
- `libcuda.so`
- `libdevice.*.bc` if your program contains device code

If you use CUDA Fortran modules, you must link in the supporting library explicitly. For example, if program `my_cublas_program.cuf` uses the `cublas` module, you must link the object file of `my_cublas_program.cuf` with the `cublas` library as follows:

```
xlcuf my_cublas_program.o -lcublas
```

Including object files not compiled with CUDA Fortran support

Your program can contain a mix of object files that are compiled with and without CUDA Fortran support. However, you must compile the object file containing the PROGRAM compilation unit with CUDA Fortran support so that the CUDA Fortran support provided by XL Fortran can be initialized.

If your program uses a `main()` function written in C, you can add a call to the following function to initialize the CUDA Fortran support provided by XL Fortran:

```
void __xlcuf_init();
```

You must place the call before any CUDA Runtime API calls in your program.

Specific compiler options and macros

These compiler options and macros are useful for CUDA Fortran programs.

Compiler options for CUDA Fortran compilation

With these options, you can customize the compilation of CUDA Fortran programs.

Table 6. Summary of CUDA Fortran specific compiler options and suboptions

Option or suboptions	Category	Purpose
The **-qcuda** option	Language element control	Enables the compiler support for CUDA Fortran.
The **-qcudaerr** option	Error checking and debugging	Controls whether the compiler automatically inserts error checking code for CUDA API calls.
The **-qpath** suboptions	Compiler customization	Specifies substitute path names for XL Fortran components.
The **-W** suboptions	Compiler customization	Passes one or more options to a specific compiler component.
The **-X** suboptions		

-qcuda (CUDA Fortran)
Category

Language element control

@PROCESS

None.

Purpose

Enables the compiler support for CUDA Fortran.

Syntax

```
►►─── -q──┬─cuda───┬──────────────────────────────────────────────►◄
          └─nocuda─┘
```
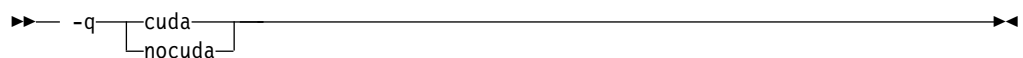
Defaults
- **-qcuda** is the default setting for the **xlcuf** invocation command, and **-qnocuda** is the default setting for other invocation commands.
- When you use **xlf_r** to invoke the compiler, **-qcuda** is the default setting for `.cuf` and `.CUF` files, and **-qnocuda** is the default setting for other file extensions.

Usage

You can specify **-qcuda** only with invocation commands that have the _r suffix; these commands ensure threadsafe compilation.

-qcuda can be used with optimization levels **-O2**, **-O3**, and **-Ofast**.

-qcuda can be used with **-qsmp=omp** but not any other suboptions of **-qsmp**. If **-qsmp=omp** is specified, only host code can contain OpenMP parallelization.

-qcuda cannot be used with **-O4**, **-O5**, **-qipa**, **-qpdf1**, **-qpdf2**, or **-qrealsize=8**.

Examples

You can use any of the following commands to compile a CUDA Fortran program:

```
xlf_r mycudaprogram1.cuf
xlf95_r -qcuda mycudaprogram2.f
xlcuf mycudaprogram3.f
xlcuf mycudaprogram4.cuf
```

-qcudaerr (CUDA Fortran)
Category

Error checking and debugging

Purpose

Controls whether the compiler automatically inserts error checking code for CUDA API calls.

Syntax

Option:

```
►►── -q─cudaerr─┬─stmts─┬──────────────────────────────────────────►◄
                ├─all───┤
                └─none──┘
```

@PROCESS:

@PROCESS CUDAERR(**STMTS** | ALL | NONE)

Defaults

-qcudaerr=stmts

Parameters

stmts Instructs the compiler to insert error checking code for the following items:
- CUDA Runtime API calls that are made in Fortran statements such as ALLOCATE, CALL, and assignment
- Synchronization points such as calls to the cudaDeviceSynchronize API function

all Instructs the compiler to insert error checking code for all CUDA Runtime API calls, including explicit calls in your program.

none Instructs the compiler not to insert error checking code for any CUDA Runtime API call.

Usage

If the inserted error checking code detects a failed CUDA Runtime API call, the compiler issues an error message, which contains the following information:
- The name of the CUDA Runtime API function that failed
- Error code
- Explanation returned by the `cudaGetErrorString` API function

The inserted error checking code does not clear the detected CUDA errors. The **-qcudaerr** option does not affect user-written error checking code.

To make the compiler exit the program when CUDA Runtime API errors are detected, you can disable runtime error recovery by either of the following ways:
- Call the `setrteopts` subroutine with `err_recovery=no`.
- Set the XLFRTEOPTS environment variable as follows:

  ```
  export XLFRTEOPTS=err_recovery=no
  ```

Examples

The example program, `out_of_bounds.cuf`, is as follows.

```
INTEGER, DEVICE, ALLOCATABLE :: device_arr(:)
INTEGER :: host_arr(10)
ALLOCATE(device_arr(5))
device_arr = host_arr
END
```

Compile it with **xlcuf** and run the executable file. The compiler issues the following message:

```
"out_of_bounds.cuf", line 4: 1525-244 API function cudaMemcpy
failed with error code 11:  invalid argument.
```

Because `out_of_bounds.cuf` contains assignment between a host array and a nonconformable device array, the assignment is translated into a `cudaMemcpy` API call. However, the API call fails because the CUDA runtime detects that `device_arr` is too small. Because **-qcudaerr=stmts** is in effect by default, the compiler checks the result of `cudaMemcpy` and displays a message containing the CUDA Runtime API error code and explanation.
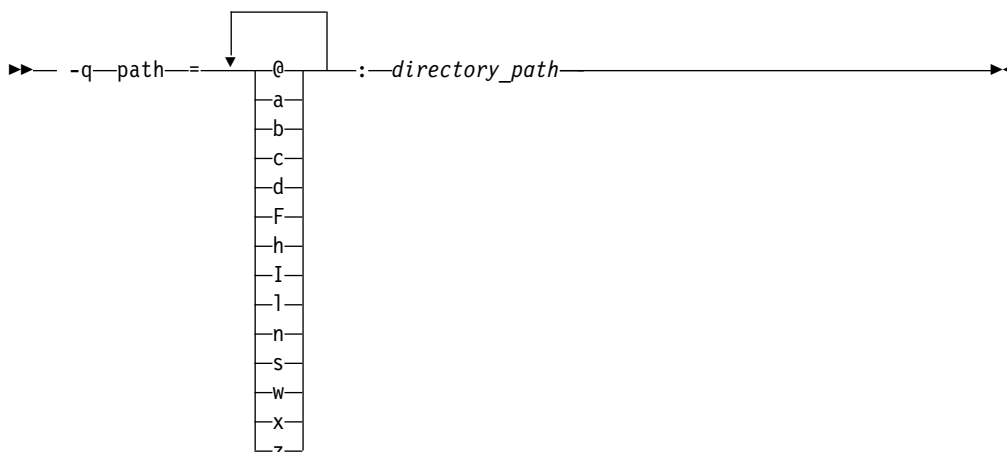
-qpath
Category

Compiler customization

Purpose

Specifies substitute path names for XL Fortran components such as the assembler, C preprocessor, and linker.

You can use this option if you want to keep multiple levels of some or all of the XL Fortran components and have the option of specifying which one you want to use.

Syntax

```
►►── -q─path──=─┬─@─┬──:─directory_path────────────────────────◄◄
                ├─a─┤
                ├─b─┤
                ├─c─┤
                ├─d─┤
                ├─F─┤
                ├─h─┤
                ├─I─┤
                ├─l─┤
                ├─n─┤
                ├─s─┤
                ├─w─┤
                ├─x─┤
                └─z─┘
```

Defaults

By default, the compiler uses the paths for compiler components defined in the configuration file.

Parameters

directory_path
   The path to the directory where the complier components are located.It must be an existing directory. It can be relative or absolute.

The following table shows the correspondence between **-qpath** parameters and the component names:

Parameter	Description	Component name
► GPU @	The PTX assembler	ptxas
a	The assembler	as
b	The low-level optimizer	xlfcode
c	The compiler front end	xlfentry
d	The disassembler	dis
F	The C preprocessor	cpp
h	The array language optimizer	xlfhot
I (uppercase i)	The high-level optimizer, compile step	ipa
l (lowercase L)	The linker	ld
► GPU n	The NVIDIA C compiler, which is used as a device linker	nvcc
► GPU s	The XL intermediate language (W-Code) splitter	partitioner
► GPU w	The XL intermediate language (W-Code) to NVVM-IR translator	wc2llvm

Parameter	Description	Component name
▶ GPU ◀ x	The NVVM-IR to PTX translator	llvm2ptx
z	The binder	bolt

Usage

The **-qpath** option overrides the **-F**, **-t**, and **-B** options.

Examples

To compile `myprogram.f` using a substitute compiler front end and linker from `/fix/FE/` and the remaining compiler components from default locations, enter the command:

```
xlf myprogram.f -qpath=cl:/fix/FE
```

To compile `myprogram.f` using a substitute compiler front end from `/fix/FE`, a substitute linker from the current directory, and the remaining compiler components from default locations, enter the command:

```
xlf95 myprogram.f -qpath=c:/fix/FE -qpath=l:.
```

-W, -X
Category

Compiler customization

@PROCESS

None.

Purpose

Passes one or more options to a component that is executed during compilation.

Syntax

```
►►── -W──┬──┬─@──┬──┬─,─option─────────────────────►◄
         │  ├─a──┤  │
         │  ├─b──┤  │
         │  ├─c──┤  │
         │  ├─d──┤  │
         │  ├─F──┤  │
         │  ├─h──┤  │
         │  ├─I──┤  │
         │  ├─L──┤  │
         │  ├─l──┤  │
         │  ├─n──┤  │
         │  ├─s──┤  │
         │  ├─w──┤  │
         │  ├─x──┤  │
         │  └─z──┘  │
```

```
                              (1)
►►──-X──┬─assembler────┬──────option────────────────────────────►◄
        ├─preprocessor─┤
        ├─linker───────┤
        ├─partitioner──┤
        ├─wc2llvm──────┤
        ├─llvm2ptx─────┤
        ├─ptxas────────┤
        └─nvcc─────────┘
```
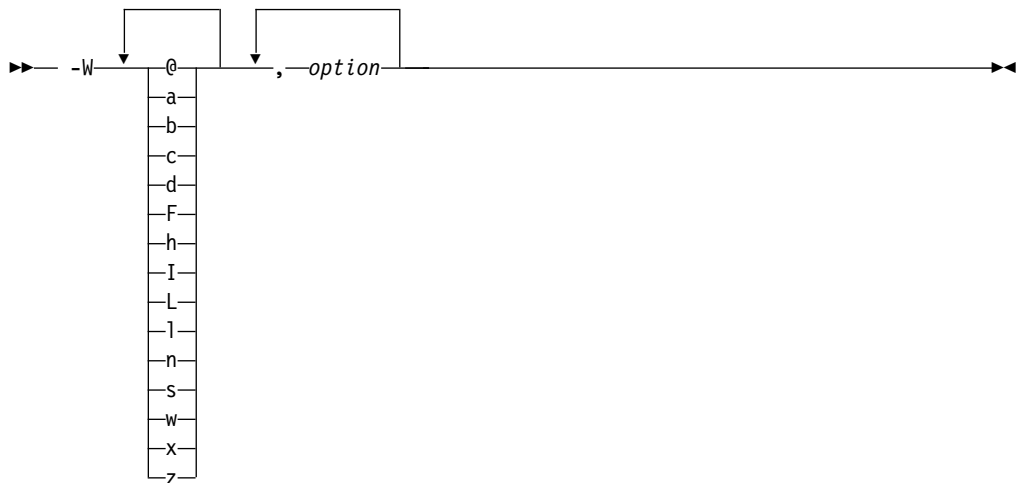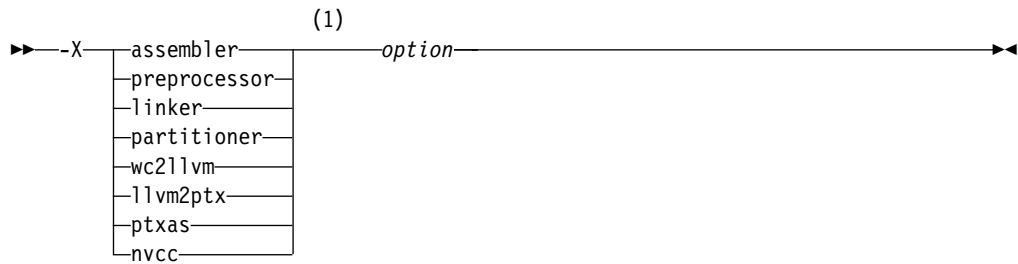
Notes:

1   You must insert at least one space before *option*.

Parameters

The following table shows the correspondence between **-X** and **-W** parameters and the component names:

Parameter of -W	Parameter of -X	Description	Component name
GPU @	ptxas	The PTX assembler	ptxas
a	assembler	The assembler	as
b		The low-level optimizer	xlfcode
c		The compiler front end	xlfentry
d		The disassembler	dis
F	preprocessor	The C preprocessor	cpp
h		The array language optimizer	xlfhot
I (uppercase i)		The high-level optimizer, compile step	ipa
L		The high-level optimizer, link step	ipa
l (lowercase L)	linker	The linker	ld
GPU n	nvcc	The NVIDIA C compiler, which is used as a device linker	nvcc
GPU s	partitioner	The XL intermediate language (W-Code) splitter	partitioner
GPU w	wc2llvm	The XL intermediate language (W-Code) to NVVM-IR translator	wc2llvm
GPU x	llvm2ptx	The NVVM-IR to PTX translator	llvm2ptx
z		The binder	bolt

option

Any option that is valid for the component to which it is being passed.

Notes: > GPU

- You can find the NVVM-IR to PTX translator options in the libNVVM API section in the CUDA Toolkit documentation at http://docs.nvidia.com/cuda/libnvvm-api/group__compilation.html under nvvmCompileProgram.
- You can get a list of the PTX assembler options by running **ptxas** from the CUDA Toolkit with **-h**.

GPU <

Usage

If you need to include a character that is special to the shell in the option string, precede the character with a backslash. For example, if you use the **-W** or **-X** option in the configuration file, you can use the escape sequence backslash comma (\,) to represent a comma in the parameter string. In the string following the **-W** option, use a comma as the separator for each option, and do not include any spaces.

You do not need the **-W** or **-X** option to pass most options to the linker **ld**; unrecognized command-line options, except **-q** options, are passed to the linker automatically. Only linker options that have the same letters as compiler options, such as **-v** or **-S**, strictly require **-W** or **-X**.

Predefined macros for CUDA Fortran support

When CUDA Fortran support is enabled, these preprocessor macros are predefined.

Table 7. Predefined macros for CUDA Fortran support

Macro	Value
_CUDA	1
_CUBLAS_V2	1
__CUDA_API_VERSION	8000

Chapter 5. Reference and limitations for CUDA Fortran support

IBM XL Fortran for Linux, V15.1.5 supports a commonly used subset of CUDA Fortran. For more information about the language extensions introduced by CUDA Fortran, see the *CUDA Fortran Programming Guide and Reference* manual downloadable from http://www.pgroup.com/doc/pgicudaforug.pdf.

The following CUDA Fortran features are not supported in IBM XL Fortran for Linux, V15.1.5:

- Calling reduction intrinsic functions, such as sum, maxval, and minval, on the host with device actual arguments
- Conditional sentinels for CUDA Fortran (!@CUF)
- CUF kernel directives
- Data transfer using the following CUDA Runtime APIs:
    - cudaMemcpyFromSymbol
    - cudaMemcpyFromSymbolAsync
    - cudaMemcpyToSymbol
    - cudaMemcpyToSymbolAsync
    - cudaMemset

    Note: You can use assignment, cudaMemcpy, or cudaMemcpyAsync instead. XL Fortran allows device global and constant module variables to appear as arguments to cudaMemcpy and cudaMemcpyAsync.
- Data transfer using the following CUDA Runtime APIs:
    - cudaMalloc3D when the first argument is a rank 3 allocatable array
    - cudaMemcpyPeer
    - cudaMemcpyPeerAsync
    - cudaMemcpy2D
    - cudaMemcpy2DAsync
    - cudaMemcpy3DAsync
    - cudaMemset2D
- Debug support

    Note: You can get basic line level debugging by compiling with **-g -qfullpath**.
- Dynamic parallelism
- PRINT and WRITE statements in device code
- Procedure definitions or interfaces that have the `attributes(host, device)` prefix

    Note: To work around this, make a copy of the procedure, and give one copy the `attributes(host)` prefix and the other copy the `attributes(device)` prefix. You must not defined the two procedures in the same compilation unit.
- Pointers with the texture attribute

    Note: The compiler automatically utilizes the texture cache for passing dummy arguments when appropriate.

- Shuffle intrinsics
- The curand module

The following limitations apply to IBM XL Fortran for Linux, V15.1.5:

- You can use CUDA Fortran with IBM XL Fortran for Linux, V15.1.5 only if the CUDA Toolkit 8.0 is installed and the compiler is configured with the location of the toolkit.
  - If you install the compiler after you install the toolkit, the compiler detects the location of the toolkit and no action is required.
  - If you install the toolkit after you install the compiler, reconfigure the compiler as described in Configuring IBM XL Fortran for Linux in the *XL Fortran Installation Guide*.

  Note: To install the CUDA Toolkit, use the Package Manager installation. The Runfile installation is currently not supported on Power processors. For instructions about Package Manager installation, see the NVIDIA CUDA Installation Guide for Linux (http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html).

- IBM XL Fortran for Linux, V15.1.5 automatically detects the GPU architecture at compiler configuration time. The GPU architecture is encoded into the compiler configuration file. No compiler options are provided to target other GPU architectures.

- Programs that use dynamic shared memory might fail due to an issue in the CUDA Toolkit 8.0. The compiler issues the following message:

```
Bitcasts between pointers of different address spaces is not legal.
Use AddrSpaceCast instead.
```

  To work around this issue, compile the affected file with the **-Xllvm2ptx -nvvm-compile-options=-opt=0** option.

Notices

Programming interfaces: Intended programming interfaces allow the customer to write programs to obtain the services of IBM XL Fortran for Linux.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA   01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1998, 2016.

This software and documentation are based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following institution for its role in this product's development: the Electrical Engineering and Computer Sciences Department at the Berkeley campus.

PRIVACY POLICY CONSIDERATIONS:

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, or to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Index

C

compilation
    CUDA Fortran programs 11
    process 5
compiler
    components 5
CUDA Fortran reference 21

H

hardware requirements
    CUDA Fortran 3

L

limitations 21
linking
    process 5

O

operating system requirements
    CUDA Fortran 3
options
    compiler 13
    CUDA Toolkit components 11
overview
    CUDA Fortran 1

P

predefined macros 19
process
    compilation 5
    linking 5

R

requirements
    hardware 3
    operating systems 3
    software 3

S

software requirements
    CUDA Fortran 3

U

unsupported CUDA Fortran features 21

IBM ®

Product Number: 5765-J10; 5725-C75

Printed in USA