


























































 Full vendor support
 Indirect, but comprehensive support, by vendor
 Vendor support, but not (yet) entirely comprehensive

 Comprehensive support, but not by vendor
 Limited, probably indirect support – but at least some
 No direct support available, but of course one could ISO-C-bind

your way through it or directly link the libraries
 C++ C++ (sometimes also C)
 Fortran Fortran

	CUDA		HIP		SYCL		OpenACC		OpenMP		Standard		Kokkos		ALPACA		etc	
	C++	Fortran	C++	Fortran	C++	Fortran	C++	Fortran	C++	Fortran	C++	Fortran	C++	Fortran	C++	Fortran	Python	Julia
NVIDIA	 1	 2	 3	 4	 5	 6	 7	 8	 9	 10	 11	 12	 13	 14	 15	 16	 17	 18
AMD	 19	 20	 21	 4	 22	 6	 23	 24	 25	 26	 27	 28	 29	 14	 30	 16	 31	 32
Intel	 33	 34	 35	 36	 37	 6	 38	 39	 40	 41	 42	 43	 44	 14	 45	 16	 46	 47

- 1: CUDA C/C++ is supported on NVIDIA GPUs through the [CUDA Toolkit](#). First released in 2007, the toolkit covers nearly all aspects of the NVIDIA platform: an API for programming (incl. language extensions), libraries, tools for profiling and debugging, compiler, management tools, and more. The current version is CUDA 12.2. Usually, when referring to *CUDA* without any additional context, the CUDA API is meant. While incorporating some Open Source components, the CUDA platform in its entirety is proprietary and closed sourced. The low-level CUDA instruction set architecture is PTX, to which higher languages like the CUDA C/C++ are translated to. PTX is compiled to SASS, the binary code executed on the device. As it is the reference for platform, the support for NVIDIA GPUs through CUDA C/C++ is very comprehensive. In addition to support through the CUDA toolkit, NVIDIA GPUs can also be [used by Clang](#), utilizing the LLVM toolchain to emit PTX code and compile it subsequently.
- 2: CUDA Fortran, a proprietary Fortran extension by NVIDIA, is supported on NVIDIA GPUs via the [NVIDIA HPC SDK \(NVHPC\)](#). NVHPC implements most features of the CUDA API in Fortran and is activated through the `-cuda` switch in the `nvfortran` compiler. The CUDA extensions for Fortran are modeled closely after the CUDA C/C++ definitions. In addition to creating explicit kernels in Fortran, CUDA Fortran also supports *cuf* kernels, a way to let the compiler generate GPU parallel code automatically. Very recently, [CUDA Fortran support was also merged into Flang](#), the LLVM-based Fortran compiler.
- 3: HIP programs can directly use NVIDIA GPUs via a CUDA backend. As HIP is strongly inspired by CUDA, the mapping is relatively straight-forward; API calls are named similarly (for example: `hipMalloc()` instead of `cudaMalloc()`) and keywords of the kernel syntax are identical. HIP also supports some CUDA libraries and creates interfaces to them (like `hipblasSaxpy()` instead of `cublasSaxpy()`). To target NVIDIA GPUs through the HIP compiler (`hipcc`), `HIP_PLATFORM=nvidia` needs to be set in the environment. In order to initially create a HIP code from CUDA, AMD offers the [HIPIFY](#) conversion tool.
- 4: No Fortran version of HIP exists; HIP is solely a C/C++ model. But AMD offers an extensive set of ready-made interfaces to the HIP API and HIP and ROCm libraries with [hipfort](#) (MIT-licensed). All interfaces implement C functionality and CUDA-like Fortran extensions, for example to write kernels, are available.
- 5: No direct support for SYCL is available by NVIDIA, but SYCL can be used on NVIDIA GPUs through multiple venues. First, SYCL can be [used through DPC++](#), an Open-Source LLVM-based compiler project [led by Intel](#). The DPC++ infrastructure is also available through Intel's commercial [oneAPI toolkit](#) (*Intel oneAPI DPC++/C++*) as [a dedicated plugin](#). Upstreaming SYCL support directly into LLVM is an [ongoing effort](#), which started in 2019. Further, SYCL can be used via [Open SYCL](#) (previously called `hipSYCL`), an independently developed SYCL implementation, using NVIDIA GPUs either through the CUDA support of LLVM or the `nvc++` compiler of NVHPC. A third popular possibility was the NVIDIA GPU support in [ComputeCpp of CodePlay](#); though [the product became unsupported in September 2023](#). In case LLVM is involved, SYCL implementations can rely on CUDA support in LLVM, which needs the CUDA toolkit available for the final compilations parts beyond PTX. In order to translate a CUDA code to SYCL, Intel offers the [SYCLomatic](#) conversion tool.
- 6: SYCL is a C++-based programming model (C++17) and by its nature does not support Fortran. Also, no pre-made bindings are available.
- 7: OpenACC C/C++ on NVIDIA GPUs is supported most extensively through the [NVIDIA HPC SDK](#). Beyond the bundled libraries, frameworks, and other models, the NVIDIA HPC SDK also features the `nvc/nvc++` compilers, in which [OpenACC support](#) can be enabled with the `-acc -gpu`. The support of OpenACC in this vendor-delivered compiler is very comprehensive, it conforms to version 2.7 of the specification. A variety of compile options are available to modify the compilation process. In addition to NVIDIA HPC SDK, good support is also available in GCC since GCC 5.0, [supporting OpenACC 2.6](#) through the `nvptx` architecture. The compiler switch to enable OpenACC in `gcc/g++` is `-fopenacc`, further options are available. Further, the [Clacc compiler](#) implements OpenACC support into the LLVM toolchain, adapting the Clang frontend. As a central design aspect, it translates OpenACC to OpenMP as part of the compilation process. OpenACC can be activated in a `Clacc-clang` via `-fopenacc`, and further compiler options exist, mostly leveraging OpenMP options. A recent study by [Jarmusch et al.](#) compared these compilers for coverage of the OpenACC 3.0 specification.
- 8: Support of OpenACC Fortran on NVIDIA GPUs is similar to OpenACC C/C++, albeit not identical. First, [NVIDIA HPC SDK](#) supports OpenACC in Fortran through the included `nvfortran` compiler, with options like for the C/C++ compilers. In addition, also [GCC supports OpenACC](#) through the `gfortran` compiler with identical compiler options to the C/C++ compilers. Further, similar to OpenACC support in LLVM for C/C++ through [Clacc](#) contributions, the LLVM frontend for Fortran, [Flang](#) (the successor of *F18*, not *classic Flang*), [supports OpenACC](#) as well. Support was initially contributed through the [Flacc project](#) and now resides in the main LLVM project. Finally, the [HPE Cray Programming Environment](#) supports [OpenACC Fortran](#); in `ftn`, OpenACC can be enabled through `-hacc`.
- 9: OpenMP in C/C++ is supported on NVIDIA GPUs ([Offloading](#)) through multiple venues, similarly to OpenACC. First, the NVIDIA HPC SDK supports [OpenMP GPU offloading](#) in both `nvc` and `nvc++`, albeit only a subset of the entire OpenMP 5.0 standard (see [the documentation for supported/unsupported features](#)). The key compiler option is `-mp`. Also in GCC, [OpenMP offloading](#) can be used to NVIDIA GPUs; the compiler switch is `-fopenmp`, with options delivered through `-offload` and `-offload-options`. GCC [currently supports OpenMP 4.5 entirely](#), while OpenMP features of 5.0, 5.1, and, 5.2 are currently being implemented. Similarly in Clang, where [OpenMP offloading to NVIDIA GPUs](#) is supported and enabled through `-fopenmp -fopenmp-targets=nvptx64`, with offload architectures selected via `--offload-arch=native` (or similar). Clang implements [nearly all OpenMP 5.0 features and most of OpenMP 5.1/5.2](#). In the HPE Cray Programming Environment, [a subset of OpenMP 5.0/5.1 is supported](#) for NVIDIA GPUs. It can be activated through `-fopenmp`. Also [AOMP](#), AMD's Clang/LLVM-based compiler, supports NVIDIA GPUs. Support of OpenMP features in the compilers was recently discussed in the [OpenMP ECP BoF 2022](#).
- 10: OpenMP in Fortran is supported on NVIDIA GPUs nearly identical to C/C++. [NVIDIA HPC SDK's nvfortran](#) implements support, [GCC's gfortran](#), [LLVM's Flang](#) (through `-mp`, and [only when Flang is compiled via Clang](#)), and also the [HPE Cray Programming Environment](#).
- 11: Standard language parallelism of C++, namely algorithms and data structures of the *parallel STL*, is supported on NVIDIA GPUs [through the nvc++ compiler of the NVIDIA HPC SDK](#). The key compiler option is `-stdpar=gpu`, which enables offloading of parallel algorithms to the GPU. Also, [AdaptiveCpp supports pSTL algorithms](#), enabled via `--acpp-stdpar`. Further, [NVIDIA GPUs can be targeted from Intel's DPC++ compiler](#), enabling usage of pSTL algorithms implemented in Intel's Open Source [oneDPL](#) (*oneAPI DPC++ Library*) on NVIDIA GPUs. Finally, a [current proposal in the LLVM community](#) aims at implementing pSTL support through an OpenMP backend.
- 12: Standard language parallelism of Fortran, mainly `do concurrent`, is supported on NVIDIA GPUs [through the nvfortran compiler of the NVIDIA HPC SDK](#). As for the C++ case, it is enabled through the `-stdpar=gpu` compiler option.
- 13: Kokkos supports NVIDIA GPUs in C++. Kokkos has [multiple backends](#) available with NVIDIA GPU support: a native CUDA C/C++ backend (using `nvcc`), an NVIDIA HPC SDK backend (using CUDA support in `nvc++`), and a Clang backend, using either Clang's CUDA support directly or [via the OpenMP offloading facilities](#) (via `clang++`).
- 14: Kokkos is a C++ programming model, but an official compatibility layer for Fortran ([Fortran Language Compatibility Layer, FLCL](#)) is available. Through this layer, GPUs can be used as supported by Kokkos C++.
- 15: Alpaka supports NVIDIA GPUs in C++ (C++17), either through the NVIDIA CUDA C/C++ compiler `nvcc` or LLVM/Clang's support of CUDA in `clang++`.
- 16: Alpaka is a C++ programming model and no ready-made Fortran support exists.

- 17: Using NVIDIA GPUs from Python code can be achieved through multiple venues. NVIDIA itself offers [CUDA Python](#), a package delivering low-level interfaces to CUDA C/C++. Typically, code is not directly written using CUDA Python, but rather CUDA Python functions as a backend for higher level models. CUDA Python is available on PyPI as [cuda-python](#). An alternative to CUDA Python from the community is [PyCUDA](#), which adds some higher-level features and functionality and comes with its own C++ base layer. PyCUDA is available on PyPI as [pycuda](#). The most well-known, higher-level abstraction is [CuPy](#), which implements primitives known from Numpy with GPU support, offers functionality for defining custom kernels, and bindings to libraries. CuPy is available on PyPI as [cupy-cuda12x](#) (for CUDA 12.x). Two packages arguably providing even higher abstractions are Numba and CuNumeric. [Numba](#) offers access to NVIDIA GPUs and features acceleration of functions through Python decorators (*functions wrapping functions*); it is available as [numba](#) on PyPI. [cuNumeric](#), a project by NVIDIA, allows to access the GPU via Numpy-inspired functions (like CuPy), but utilizes the [Legate library](#) to transparently scale to multiple GPUs.
- 18: Using NVIDIA GPUs in Julia is possible through the community-supported [CUDA.jl](#) package that enables both low-level kernel programming in Julia as well as high-level array-based programming.
- 19: While CUDA is generally not directly supported on AMD GPUs, it can be translated to HIP through AMD's [HIPiFY](#). Using `hipcc` and `HIP_PLATFORM=amd` in the environment, CUDA-to-HIP-translated code can be executed. In addition, a third-party, open source library/framework is currently being developed to execute CUDA code on AMD GPUs, without source code modifications, called [ZLUDA](#).
- 20: No direct support for CUDA Fortran on AMD GPUs is available, but AMD offers a source-to-source translator, [GPUFORT](#), to convert some CUDA Fortran to either Fortran with OpenMP (via [AOMP](#)) or Fortran with HIP bindings and extracted C kernels (via [hipfort](#)). As stated in the project repository, the covered functionality is [driven by use-case requirements](#); the last commit is two years old.
- 21: [HIP C++](#) is the *native* programming model for AMD GPUs and, as such, fully supports the devices. It is part of AMD's GPU-targeted [ROCm platform](#), which includes compilers, libraries, tool, and drivers and mostly consists of Open Source Software. HIP code can be compiled with `hipcc`, utilizing the correct environment variables (like `HIP_PLATFORM=amd`) and compiler options (like `-foffload-arch=gfx90a`). `hipcc` is a *compiler driver* (wrapper script) which assembles the correct compilation string, finally calling [AMD's Clang compiler](#) to generate host/device code (using the [AMDGPU backend](#)).
- 22: No direct support for SYCL is available by AMD for their GPU devices. But like for the NVIDIA ecosystem, SYCL C++ can be used on AMD GPUs through third-party software. First, [Open SYCL](#) (previously [hipSYCL](#)) supports AMD GPUs, relying on HIP/ROCm support in Clang. All available [internal compilation models](#) can target AMD GPUs. Second, also AMD GPUs can be targeted through both [DPC++](#), Intel's LLVM-based Open Source compiler, and the commercial version included in the [oneAPI toolkit](#) (via an [AMD ROCm plugin](#)). In comparison to SYCL support for CUDA, no conversion tool like SYCLomatic exists.
- 23: OpenACC C/C++ is not supported by AMD itself, but third-party support is available for AMD GPUs through GCC or Clacc (similarly to their support of OpenACC C/C++ for NVDI GPUs). In [GCC](#), [OpenACC support](#) can be activated through `-fopenacc`, and further specified for AMD GPUs with, for example, `-foffload=amdgc-n-amdhsa=-march=gfx906`. [Clacc also supports OpenACC C/C++ on AMD GPUs](#) by translating OpenACC to OpenMP and using LLVM's AMD support. The enabling compiler switch is `-fopenacc`, and AMD GPU targets can be further specified by, for example, `-fopenmp-targets=amdgc-n-amd-amdhsa`. [Intel's OpenACC to OpenMP source-to-source translator](#) can also be used for AMD's platform.
- 24: No native support for OpenACC on AMD GPUs for Fortran is available, but AMD supplies [GPUFORT](#), a research project to source-to-source translate OpenACC Fortran to either Fortran with added OpenMP or Fortran with HIP bindings and extracted C kernels (using [hipfort](#)). The covered functionality of GPUFORT is driven by use-case requirements, the last commit is two years old. Support for OpenACC Fortran is also available by the community through [GCC \(gfortran\)](#) and upcoming in [LLVM \(Flacc\)](#). Also the [HPE Cray Programming Environment supports OpenACC Fortran](#) on AMD GPUs. In addition, the [translator tool to convert OpenACC source to OpenMP source](#) by Intel can be used.
- 25: AMD offers [AOMP](#), a dedicated, Clang-based compiler for using OpenMP C/C++ on AMD GPUs (*offloading*). AOMP is usually shipped with ROCm. The compiler [supports most OpenMP 4.5 and some OpenMP 5.0 features](#). Since the compiler is Clang-based, the usual Clang compiler options apply (`-fopenmp` to enable OpenMP parsing, and others). Also in the upstream Clang compiler, [AMD GPUs can be targeted through OpenMP](#); as outlined for NVIDIA GPUs, the support for OpenMP 5.0 is nearly complete, and support for OpenMP 5.1/5.2 is comprehensive. In addition, the [HPE Cray Programming Environment](#) supports OpenMP on AMD GPUs.
- 26: Through [AOMP](#), AMD supports OpenMP offloading to AMD GPUs in Fortran, using the `fLang` executable and Clang-typical compiler options (foremost `-fopenmp`). Support for AMD GPUs is also available through the [HPE Cray Programming Environment](#).
- 27: AMD does not yet provide production-grade support for Standard-language parallelism in C++ for their GPUs. Currently under development is [roc-stdpar](#) (ROCm Standard Parallelism Runtime Implementation), which aims to supply pSTL algorithms on the GPU and [merge the implementation with upstream LLVM](#). Support for GPU-parallel algorithms is enabled with `-stdpar`. An [alternative proposal in the LLVM community](#) aims to support the pSTL via an OpenMP backend. Also [AdaptiveCpp supports C++ parallel algorithms](#) via a `--acpp-stdpar` switch. Intel provides the Open Source [oneDPL \(oneAPI DPC++ Library\)](#) which [implements pSTL algorithms](#) through the DPC++ compiler (see also [C++ Standard Parallelism for Intel GPUs](#)). DPC++ has [experimental support for AMD GPUs](#).
- 28: Recently, standard parallelism in Fortran is supported on AMD GPUs via AMD's *Next Gen Fortran Compiler* (based on the new LLVM Flang). Support is being improved, a summary is [available on GitHub](#). Also in [HPE's Cray Fortran compiler](#), Standard parallelism is supported.
- 29: [Kokkos](#) supports AMD GPUs in C++ mainly through the HIP/ROCm backend. Also, an OpenMP offloading backend is available.
- 30: [Alpaka](#) supports AMD GPUs in C++ through HIP or through an OpenMP backend.
- 31: AMD supports GPU programming with Python via [HIP Python](#), providing low-level bindings for HIP. [CuPy](#) experimentally supports AMD GPUs/ROCm. The package can be found on PyPI as `cupy-rocmm-5-0`. Numba once had upstream [support for AMD GPUs](#), but it is [not maintained anymore](#); meanwhile, an [experimental HIP backend](#) for Numba exists maintained by AMD. Low-level bindings from Python to HIP exist, for example [PyHIP](#) (available as `pyhip-interface` on PyPI). Bindings to OpenCL also exist ([PyOpenCL](#)).
- 32: Using AMD GPUs in Julia is possible through the community supported [AMDGPU.jl](#) package.
- 33: Intel itself does not support CUDA C/C++ on their GPUs. They offer [SYCLomatic](#), though, an Open Source tool to translate CUDA code to SYCL code, allowing it to run on Intel GPUs. The commercial variant of SYCLomatic is called the [DPC++ Compatibility Tool](#) and bundled with oneAPI toolkit. The community project [chipStar](#) (previously called CHIP-SPV, recently released a 1.0 version) allows to target Intel GPUs from CUDA C/C++ code by using the CUDA support in Clang. [chipStar](#) delivers a [Clang-wrapper, cucc](#), which replaces calls to `nvcc`.
- 34: No direct support exists for CUDA Fortran on Intel GPUs. A simple example to bind SYCL to a (CUDA) Fortran program (via ISO C BINDING) can be [found on GitHub](#).
- 35: No native support for HIP C++ on Intel GPUs exists. The Open Source third-party project [chipStar](#) (previously called CHIP-SPV), though, supports [HIP on Intel GPUs](#) by mapping it to OpenCL or Intel's Level Zero runtime. The compiler uses an LLVM-based toolchain and relies on its HIP and SPIR-V functionality.
- 36: HIP for Fortran does not exist, and also no translation efforts for Intel GPUs.
- 37: [SYCL](#) is a C++17-based standard and selected by Intel as the prime programming model for Intel GPUs. Intel implements SYCL support for their GPUs [via DPC++](#), an LLVM-based compiler toolchain. Currently, Intel maintains an own fork of LLVM, but [plans to upstream the changes](#) to the main LLVM repository. Based on DPC++, Intel releases a [commercial Intel oneAPI DPC++ compiler](#) as part of the [oneAPI toolkit](#). The third-party project Open SYCL also supports Intel GPUs, by leveraging/creating LLVM support (either SPIR-V or Level Zero). A previous solution for targeting Intel GPUs from SYCL was [ComputeCpp of CodePlay](#). The project became unsupported in September 2023 (in favor of implementations to the DPC++ project).
- 38: No direct support for OpenACC C/C++ is available for Intel GPUs. Intel offers a Python-based tool to translate source files with OpenACC C/C++ to OpenMP C/C++, the [Application Migration Tool for OpenACC to OpenMP API](#).
- 39: Also for OpenACC Fortran, no direct support is available for Intel GPUs. Intel's [source-to-source translation tool from OpenACC to OpenMP](#) also supports Fortran, though.
- 40: OpenMP is a second key programming model for Intel GPUs and [well-supported by Intel](#). For C++, the support is built into the commercial version of DPC++/C++, [Intel oneAPI DPC++/C++](#). All [OpenMP 4.5 and most OpenMP 5.0 and 5.1 features are supported](#). OpenMP can be enabled through the `-qopenmp` compiler option of `icpx`; a suitable offloading target can be given via `-fopenmp-targets=spir64`.
- 41: OpenMP in Fortran is Intel's main selected route to bring Fortran applications to their GPUs. OpenMP offloading in Fortran is supported through [Intel's Fortran Compiler icx](#) (the new LLVM-based version, not the *Fortran Compiler Classic*), part of the oneAPI HPC Toolkit. Similarly to C++, OpenMP offloading can be enabled through a combination of `-qopenmp` and `-fopenmp-targets=spir64`.
- 42: Intel supports C++ standard parallelism (pSTL) through the Open Source [oneDPL](#) (oneAPI DPC++ Library), also available as part of the oneAPI toolkit. It [implements the pSTL](#) on top of the DPC++ compiler, algorithms, data structures, and policies live in the `oneapi::dpl::` namespace. In addition, [AdaptiveCPP supports C++ parallel algorithms](#) on Intel GPUs, enabled via the `--acpp-stdpar` compiler option.
- 43: Standard language parallelism of Fortran is supported by Intel on their GPUs through the Intel Fortran Compiler `icx` (the new, LLVM-based compiler, not the *Classic* version), part of the oneAPI HPC

toolkit. In the [oneAPI update 2022.1](#), the `do_concurrent` support was added and extended in further releases. It can be used via the `-qopenmp` compiler option together with `-fopenmp-target-do-concurrent` and `-fopenmp-targets=spir64`.

- [44](#): No direct support by Intel for Kokkos is available, but [Kokkos](#) supports Intel GPUs through an experimental SYCL backend.
- [45](#): Since [v.0.9.0](#), [Alpaka](#) contains experimental SYCL support with which Intel GPUs can be targeted. Also, Alpaka can fall back to an OpenMP backend.
- [46](#): Intel GPUs can be used from Python through three notable packages. First, Intel's [Data Parallel Control \(dpctl\)](#) implements low-level Python bindings to SYCL functionality. It is available on PyPI as `dpctl`. Second, a higher level, Intel's [Data-parallel Extension to Numba \(numba-dpex\)](#) supplies an extension to the JIT functionality of Numba to support Intel GPUs. It is available from Anaconda as `numba-dpex`. Finally, and arguably highest level, Intel's [Data Parallel Extension for Numpy \(dpnp\)](#) builds up on the Numpy API and extends some functions with Intel GPU support. It is available on PyPI as `dpnp`, although latest versions appear to be available [only on GitHub](#).
- [47](#): Using Intel GPUs in Julia is possible through the community supported [oneAPI.jl](#) package