SC23
Denver, CO | i am hpc.

Many Cores, Many Models

GPU Programming Model vs. Vendor Compatibility Overview

P3HPC Workshop

Andreas Herten, Jülich Supercomputing Centre, Forschungszentrum Jülich

Overview

State of the GPUnion

- GPUs: prevalent in largest HPC installations, enablers of Exascale
  Top500 June 23: 70 % of FLOP/S by GPUs, > 100 000 GPUs in Frontier+Aurora
- 3 vendors (AMD, Intel, NVIDIA), each with *native* programming model (HIP, SYCL, CUDA)
- Further models, partly from community: OpenMP, OpenACC; Kokkos, RAJA, Alpaka; Standard Parallelism; *Python and more!*
- Major languages: C/C++, Fortran
- Plethora of possibilities: $3 \times 9 \times 2 = 54 \rightarrow$ What to choose?

State of the GPUnion

- GPUs: prevalent in largest HPC installations, enablers of Exascale
  Top500 June 23: 70 % of FLOP/s by GPUs, > 100 000 GPUs in Frontier+Aurora
- 3 vendors (AMD, Intel, NVIDIA), each with *native* programming model (HIP, SYCL, CUDA)
- Further models, partly from community: OpenMP, OpenACC; Kokkos, RAJA, Alpaka; Standard Parallelism; *Python and more!*
- Major languages: C/C++, Fortran
- Plethora of possibilities: $3 \times 9 \times 2 = 54$ → **What to choose?**

State of the GPUnion

- GPUs: prevalent in largest HPC installations, enablers of Exascale
  Top500 June 23: 70 % of FLOP/S by GPUs, > 100 000 GPUs in Frontier+Aurora
- 3 vendors (AMD, Intel, NVIDIA), each with *native* programming model (HIP, SYCL, CUDA)
- Further models, partly from community: OpenMP, OpenACC; Kokkos, RAJA, Alpaka; Standard Parallelism; *Python and more!*
- Major languages: C/C++, Fortran
- Plethora of possibilities: 3 × 9 × 2 = 54 → **What to choose?**

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

State of the GPUnion

- GPUs: prevalent in largest HPC installations, enablers of Exascale
  Top500 June 23: 70 % of FLOP/S by GPUs, > 100 000 GPUs in Frontier+Aurora
- 3 vendors (AMD, Intel, NVIDIA), each with *native* programming model (HIP, SYCL, CUDA)
- Further models, partly from community: OpenMP, OpenACC; Kokkos, RAJA, Alpaka; Standard Parallelism; *Python and more!*
- Major languages: C/C++, Fortran
- Plethora of possibilities: $3 \times 9 \times 2 = 54$ → **What to choose?**

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

The Table *(Split)*

	CUDA		HIP		SYCL	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 1	● 2	◒ 3	★ / 4	▲ 5	/ 6
AMD	◒ 18	★ 19	● 20	★ / 4	▲ 21	/ 6
Intel	◒ ▲ 31	/ 32	▲ 33	/ 34	● 35	/ 6

	OpenACC		OpenMP		Standard	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 7	● 8	■ ▲ 9	■ ● 10	● 11	● 12
AMD	▲ 22	▲ ★ 23	● ▲ 24	● 25	▲ ■ ★ 26	/ 27
Intel	★ 36	★ 37	● 38	● 39	● ■ 40	● 41

	Kokkos		ALPAKA		
	C++	Fortran	C++	Fortran	Python
NVIDIA	▲ 13	★ 14	▲ 15	/ 16	● ▲ 17
AMD	▲ 28	★ 14	▲ 29	/ 16	★ 30
Intel	▲ 42	★ 14	▲ 43	/ 16	■ 44

- State: Sep 2023 *evolving*
- Categories

●	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
C++	C++ (sometimes also C)
Fortran	Fortran

Highlight: CUDA

	CUDA		HIP		SYCL	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 1	● 2	◒ 3	★ / 4	▲ 5	/ 6
AMD	◒ 18	★ 19	● 20	★ / 4	▲ 21	/ 6
Intel	◒ ▲ 31	/ 32	▲ 33	/ 34	● 35	/ 6

	OpenACC		OpenMP		Standard	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 7	● 8	■ ▲ 9	■ ● 10	● 11	● 12
AMD	▲ 22	▲ ★ 23	● ▲ 24	● 25	▲ ■ ★ 26	/ 27
Intel	★ 36	★ 37	● 38	● 39	● ■ 40	● 41

	Kokkos		ALPAKA		Python
	C++	Fortran	C++	Fortran	
NVIDIA	▲ 13	★ 14	▲ 15	/ 16	● ▲ 17
AMD	▲ 28	★ 14	▲ 29	/ 16	★ 30
Intel	▲ 42	★ 14	▲ 43	/ 16	■ 44

CUDA

NVIDIA	*Native* model; C/C++, Fortran; proprietary Also: Clang
AMD	CUDA C++ not directly supported, but conversion to HIP via *HIPIFY*; Fortran via S2S translator GPUFORT (incl. hipfort)
Intel	CUDA C++ not directly supported, but via conversion to SYCL via SYCLomatic (*DPC++ Compatibility Tool*) Also: chipStar (via cuspv via Clang)

Highlight: Standard

	CUDA		HIP		SYCL	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 1	● 2	◒ 3	★ ⁄ 4	▲ 5	⁄ 6
AMD	◒ 18	★ 19	● 20	★ ⁄ 4	▲ 21	⁄ 6
Intel	◒ ▲ 31	⁄ 32	▲ 33	⁄ 34	● 35	⁄ 6

	OpenACC		OpenMP		**Standard**	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 7	● 8	■ ▲ 9	■ ● 10	● 11	● 12
AMD	▲ 22	▲ ★ 23	● ▲ 24	● 25	▲ ■ ★ 26	⁄ 27
Intel	★ 36	★ 37	● 38	● 39	● ■ 40	● 41

	Kokkos		ALPAKA		
	C++	Fortran	C++	Fortran	Python
NVIDIA	▲ 13	★ 14	▲ 15	⁄ 16	● ▲ 17
AMD	▲ 28	★ 14	▲ 29	⁄ 16	★ 30
Intel	▲ 42	★ 14	▲ 43	⁄ 16	■ 44

Standard Language Parallelism

C++: pSTL; Fortran: `do concurrent`

- NVIDIA ▶ Enable through `-stdpar=gpu` in NVHPC's `nvc++` and `nvfortran`
  Also (C++): Open SYCL; WIP Intel's oneDPL, via DPC++ compiler; WIP, pSTL in LLVM via OpenMP
- AMD ▶ No production-level support, but WIP on *roc-stdpar* (LLVM, C++)
  Also (C++): Open SYCL; WIP oneDPL; WIP LLVM via OpenMP
- Intel ▶ C++: oneDPL; Fortran: enable through `-fopenmp-target-do-concurrent` in `ifx`
  Also (C++): Open SYCL

All Descriptions

	CUDA		HIP		SYCL	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 1	● 2	◡ 3	★ / 4	▲ 5	/ 6
AMD	◡ 18	★ 19	● 20	★ / 4	▲ 21	/ 6
Intel	◡ ▲ 31	/ 32	▲ 33	/ 34	● 35	/ 6

	OpenACC		OpenMP		Standard	
	C++	Fortran	C++	Fortran	C++	Fortran
NVIDIA	● 7	● 8	■ ▲ 9	■ ● 10	● 11	● 12
AMD	▲ 22	▲ ★ 23	● ▲ 24	● 25	▲ ■ ★ 26	/ 27
Intel	★ 36	★ 37	● 38	● 39	● ■ 40	● 41

	Kokkos		ALPAKA		
	C++	Fortran	C++	Fortran	Python
NVIDIA	▲ 13	★ 14	▲ 15	/ 16	● ▲ 17
AMD	▲ 28	★ 14	▲ 29	/ 16	★ 30
Intel	▲ 42	★ 14	▲ 43	/ 16	■ 44

All The Details

- See appendix for all descriptions and references
- Or: the paper!

Discussions

Model Selection	Prevalence, representative cross-section Notable omission: RAJA (→Kokkos), OpenCL (rare; SYCL), HPX, …
Performance	Complicated! Hard to do right/fair on this scale! But: **this workshop series**
Topicality	Rapidly evolving field, especially if OSS (/LLVM), already some new developments since paper submission But: also silent deprecation (GPUFORT)?
Assessments	As objective as possible, but details more involved Example: NVIDIA OpenACC C++ 🟢 vs. OpenMP C++ 🟧; validation suites helpful

Conclusion

- Overview of support for GPU programming models for HPC languages on GPU platforms

(CUDA, HIP, SYCL, OpenACC, OpenMP, Standard, Kokkos, Alpaka, Python) $\otimes$ (C++, Fortran) $\otimes$ (NVIDIA, AMD, Intel)

- Detailed descriptions of support, assessment of status
- Table for novel GPU developers and seasoned experts
- A lot has happened, ecosystem is much more diverse right now

Conclusion

- Overview of support for GPU programming models for HPC languages on GPU platforms

(CUDA, HIP, SYCL, OpenACC, OpenMP, Standard, Kokkos, Alpaka, Python) $\otimes$ (C++, Fortran) $\otimes$ (NVIDIA, AMD, Intel)

- Detailed descriptions of support, assessment of status
- Table for novel GPU developers and seasoned experts
- A lot has happened, ecosystem is much more diverse right now
- Data for future beyond paper
  - Raw data, scripts on GitHub, welcoming issues/pull requests
    `github.com/AndiH/gpu-lang-compat/`
  - Generated website for *living* document
    `x-dev.pages.jsc.fz-juelich.de/models/`

Conclusion

- Overview of support for GPU programming models for HPC languages on GPU platforms

(CUDA, HIP, SYCL, OpenACC, OpenMP, Standard, Kokkos, Alpaka, Python) $\otimes$ (C++, Fortran) $\otimes$ (NVIDIA, AMD, Intel)

- Detailed descriptions of support, assessment of status
- Table for novel GPU developers and seasoned experts
- A lot has happened, ecosystem is much more diverse right now
- Data for future beyond paper
  - Raw data, scripts on GitHub, welcoming issues/pull requests
    `github.com/AndiH/gpu-lang-compat/`
  - Generated website for *living* document
    `x-dev.pages.jsc.fz-juelich.de/models/`

Thanks to all people providing input, discussions

Conclusion

- Overview of support for GPU programming models for HPC languages on GPU platforms

(CUDA, HIP, SYCL, OpenACC, OpenMP, Standard, Kokkos, Alpaka, Python) ⊗ (C++, Fortran) ⊗ (NVIDIA, AMD, Intel)

- Detailed descriptions of support, assessment of status
- Table for novel GPU developers and seasoned experts
- A lot has happened, ecosystem is much more diverse right now
- Data for future beyond paper
  - Raw data, scripts on GitHub, welcoming issues/pull requests
    `github.com/AndiH/gpu-lang-compat/`
  - Generated website for *living* document
    `x-dev.pages.jsc.fz-juelich.de/models/`

Thanks to all people providing input, discussic

Thank you for your attention!
a.herten@fz-juelich.de
@andih@mastodon.social

JÜLICH Forschungszentrum

Appendix

Appendix

Descriptions

Detailed Description I

1 **NVIDIA • CUDA • C++:** 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. [1]

2 **NVIDIA • CUDA • Fortran:** CUDA Fortran, a proprietary Fortran extension by NVIDIA, is supported on NVIDIA GPUs via the NVIDIA HPC SDK (*NVHPC*). NVHPC implements most features

Detailed Description II

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. [2]

3 **NVIDIA • HIP • C++:** 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. [3]

Detailed Description III

4 **NVIDIA, AMD • HIP • Fortran:** 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. [4]

5 **NVIDIA • SYCL • C++:** 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

Detailed Description IV

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. [5, 6]

6 **NVIDIA, AMD, Intel • SYCL • Fortran:** 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]

7 **NVIDIA • OpenACC • C++:** 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.

Detailed Description V

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. [8–11]

8 **NVIDIA • OpenACC • Fortran:** 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

Detailed Description VI

resides in the main LLVM project. Finally, the HPE Cray Programming Environment supports OpenACC Fortran; in `ftn-hacc`. [8, 9, 12]

9 **NVIDIA • OpenMP • C++:** 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 `-foffload` and `-foffload-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

Detailed Description VII

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. [8, 13–15]

10 **NVIDIA • OpenMP • Fortran:** 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. [8, 13, 15, 16]

11 **NVIDIA • Standard • C++:** 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, currently Open SYCL is in the process of implementing support for pSTL algorithms, enabled via `--hipsycl-stdpar`. Further, NVIDIA GPUs can be targeted from Intel's DPC++ compiler, enabling usage of pSTL algorithms implemented in Intel's Open Source

Detailed Description VIII

oneDPL (*oneAPI DPC++ Library*) on NVIDIA GPUs. Finally, a current proposal in the LLVM community aims at implementing pSTL support through an OpenMP backend. [6, 8, 17]

12 **NVIDIA • Standard • Fortran:** 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. [8]

13 **NVIDIA • Kokkos • C++:** 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++`). [18]

14 **NVIDIA, AMD, Intel • Kokkos • Fortran:** 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++. [18]

Detailed Description IX

15 **NVIDIA • ALPAKA • C++:** 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++`. [19]

16 **NVIDIA, AMD, Intel • ALPAKA • Fortran:** Alpaka is a C++ programming model and no ready-made Fortran support exists. [19]

17 **NVIDIA • etc • Python:** 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

Detailed Description X

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. [20–24]

18 **AMD • CUDA • C++:** While CUDA is 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. [3]

19 **AMD • CUDA • Fortran:** 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. [25]

Detailed Description XI

20 **AMD • HIP • C++:** 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 `--offload-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). [3]

21 **AMD • SYCL • C++:** 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

Detailed Description XII

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. [5, 6]

22 **AMD • OpenACC • C++:** 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 NVIDI GPUS). In GCC, OpenACC support can be activated through `-fopenacc`, and further specified for AMD GPUs with, for example, `-foffload=amdgcn-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=amdgcn-amd-amdhsa`. Intel's OpenACC to OpenMP source-to-source translator can also be used for AMD's platform. [9, 10]

23 **AMD • OpenACC • Fortran:** 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

Detailed Description XIII

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. [9, 12, 25]

24 **AMD • OpenMP • C++:** 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

Detailed Description XIV

comprehensive. In addition, the HPE Cray Programming Environment supports OpenMP on AMD GPUs. [15, 26, 27]

25 **AMD • OpenMP • Fortran:** 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. [15, 26]

26 **AMD • Standard • C++:** 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 Open SYCL is in the process of creating support for C++ parallel algorithms via a `--hipsycl-stdpar` switch. By using Open SYCL's backends,

Detailed Description XV

also AMD GPUs are supported. 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. [6, 17, 28]

27 **AMD • Standard • Fortran:** There is no (known) way to launch Standard-based parallel algorithms in Fortran on AMD GPUs.

28 **AMD • Kokkos • C++:** Kokkos supports AMD GPUs in C++ mainly through the HIP/ROCm backend. Also, an OpenMP offloading backend is available. [18]

29 **AMD • ALPAKA • C++:** Alpaka supports AMD GPUs in C++ through HIP or through an OpenMP backend. [19]

30 **AMD • etc • Python:** AMD does not officially support GPU programming with Python, but third-party solutions are available. CuPy experimentally supports AMD GPUs/ROCm. The package can be found on PyPI as `cupy-rocm-5-0`. Numba once had support for AMD GPUs, but

Detailed Description XVI

it is not maintained anymore. Low-level bindings from Python to HIP exist, for example PyHIP (available as `pyhip-interface` on PyPI). Bindings to OpenCL also exist (PyOpenCL). [20]

31 **Intel • CUDA • C++:** 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, `cuspv`, which replaces calls to `nvcc`. Also ZLUDA exists, which implements CUDA support for Intel GPUs; it is not maintained anymore, though. [29–31]

32 **Intel • CUDA • Fortran:** 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.

Detailed Description XVII

33 **Intel • HIP • C++:** 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. [30]

34 **Intel • HIP • Fortran:** HIP for Fortran does not exist, and also no translation efforts for Intel GPUs.

35 **Intel • SYCL • C++:** 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

Detailed Description XVIII

became unsupported in September 2023 (in favor of implementations to the DPC++ project). [5, 6, 31]

36 **Intel • OpenACC • C++:** 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*. [32]

37 **Intel • OpenACC • Fortran:** 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. [32]

38 **Intel • OpenMP • C++:** 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`. [31]

Detailed Description XIX

39 **Intel • OpenMP • Fortran:** 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 `ifx` (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`. [31]

40 **Intel • Standard • C++:** 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, Open SYCL is current adding support for C++ parallel algorithms, to be enabled via the `--hipsycl-stdpar` compiler option. [17]

41 **Intel • Standard • Fortran:** Standard language parallelism of Fortran is supported by Intel on their GPUs through the Intel Fortran Compiler `ifx` (the new, LLVM-based compiler, not the *Classic* version), part of the oneAPI HPC toolkit. In the oneAPI update 2022.1, the do

Detailed Description XX

`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`. [31]

42 **Intel • Kokkos • C++:** No direct support by Intel for Kokkos is available, but Kokkos supports Intel GPUs through an experimental SYCL backend. [18]

43 **Intel • ALPAKA • C++:** 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.

44 **Intel • etc • Python:** 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

Detailed Description XXI

some functions with Intel GPU support. It is available on PyPI as dpnp, although latest versions appear to be available only on GitHub. [33–35]

Appendix

References

References I

[1] NVIDIA. *CUDA Toolkit*. 2023. URL: `https://developer.nvidia.com/cuda-toolkit` (page 19).

[2] NVIDIA. *CUDA Fortran*. 2023. URL: `https://developer.nvidia.com/cuda-fortran` (visited on 10/26/2023) (page 20).

[3] AMD. *HIP*. 2023. URL: `https://rocm.docs.amd.com/projects/HIP/en/latest/` (pages 20, 28, 29).

[4] AMD. *hipfort*. 2023. URL: `https://rocm.docs.amd.com/projects/hipfort/en/latest/` (page 21).

[5] Intel and Contributors. *oneAPI DPC++ Compiler*. LLVM-fork with DPC++ support by Intel. 2023. URL: `https://github.com/intel/llvm` (pages 22, 30, 36).

References II

[6] Aksel Alpay et al. “Exploring the possibility of a hipSYCL-based implementation of oneAPI.” In: *International Workshop on OpenCL*. ACM, May 2022. DOI: 10.1145/3529538.3530005. URL: https://doi.org/10.1145/3529538.3530005 (pages 22, 26, 30, 33, 36).

[7] Khronos Group. *SYCL*. 2023. URL: https://www.khronos.org/sycl/ (page 22).

[8] NVIDIA. *NVIDIA HPC SDK*. 2023. URL: https://developer.nvidia.com/hpc-sdk (pages 23–26).

[9] GCC. *GCC OpenACC*. 2023. URL: https://gcc.gnu.org/wiki/OpenACC (pages 23, 24, 30, 31).

[10] Joel E. Denny, Seyong Lee, and Jeffrey S. Vetter. “CLACC: Translating OpenACC to OpenMP in Clang.” In: *2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*. 2018, pp. 18–29. DOI: 10.1109/LLVM-HPC.2018.8639349 (pages 23, 30).

JÜLICH
Forschungszentrum

References III

[11] Aaron Jarmusch et al. “Analysis of Validating and Verifying OpenACC Compilers 3.0 and Above.” In: *2022 Workshop on Accelerator Programming Using Directives (WACCPD)*. 2022, pp. 1–10. DOI: 10.1109/WACCPD56842.2022.00006 (page 23).

[12] Valentin Clement and Jeffrey S. Vetter. “Flacc: Towards OpenACC support for Fortran in the LLVM Ecosystem.” In: *2021 IEEE/ACM 7th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*. 2021, pp. 12–19. DOI: 10.1109/LLVMHPC54804.2021.00007 (pages 24, 31).

[13] GCC Developers. *GCC OpenMP*. 2023. URL: https://gcc.gnu.org/wiki/openmp (page 25).

[14] LLVM/Clang Developers. *Clang OpenMP*. 2023. URL: https://clang.llvm.org/docs/OpenMPSupport.html (page 25).

References IV

[15] HPE. *HPE Cray Programming Environment*. 2023. URL: https://www.hpe.com/psnow/doc/a50002303enw (pages 25, 32).

[16] LLVM/Flang. *Flang*. 2023. URL: https://flang.llvm.org/ (page 25).

[17] Intel. *oneDPL*. 2023. URL: https://oneapi-src.github.io/oneDPL/index.html (pages 26, 33, 37).

[18] Christian R. Trott et al. “Kokkos 3: Programming Model Extensions for the Exascale Era.” In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2022), pp. 805–817. DOI: 10.1109/TPDS.2021.3097283 (pages 26, 33, 38).

[19] A. Matthes et al. “Tuning and optimization for a variety of many-core architectures without changing a single line of implementation code using the Alpaka library.” In: June 2017. arXiv: 1706.10086. URL: https://arxiv.org/abs/1706.10086 (pages 27, 33).

References V

[20] NVIDIA. *CUDA Python*. 2023. URL: https://nvidia.github.io/cuda-python/index.html (pages 28, 34).

[21] Andreas Kloeckner et al. *PyCUDA*. Version v2022.2.2. July 2023. DOI: 10.5281/zenodo.8121901. URL: https://doi.org/10.5281/zenodo.8121901 (page 28).

[22] Ryosuke Okuta et al. “CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations.” In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf (page 28).

[23] Siu Kwan Lam et al. *numba/numba: Version 0.57.1*. Version 0.57.1. June 2023. DOI: 10.5281/zenodo.8087361. URL: https://doi.org/10.5281/zenodo.8087361 (page 28).

JÜLICH Forschungszentrum

References VI

[24] NVIDIA. *cuNumeric*. 2023. URL: https://developer.nvidia.com/cunumeric (page 28).

[25] AMD. *GPUFORT*. 2023. URL: https://github.com/ROCmSoftwarePlatform/gpufort (pages 28, 31).

[26] AMD. *AOMP*. 2023. URL: https://github.com/ROCm-Developer-Tools/aomp (page 32).

[27] ECP Exascale Computing Project. *OpenMP Roadmap for Accelerators Across DOE Pre-Exascale/Exascale Machines*. 2022. URL: https://www.openmp.org/wp-content/uploads/2022_ECP_Community_BoF_Days-OpenMP_RoadMap_BoF.pdf (page 32).

References VII

[28] AMD. *roc-stdpar*. 2023. URL: https://github.com/ROCmSoftwarePlatform/roc-stdpar (page 33).

[29] Intel. *SYCLomatic*. 2023. URL: https://github.com/oneapi-src/SYCLomatic (page 34).

[30] Jisheng Zhao et al. "HIPLZ: Enabling Performance Portability for Exascale Systems." In: *Euro-Par 2022: Parallel Processing Workshops*. Ed. by Jeremy Singer et al. Cham: Springer Nature Switzerland, 2023, pp. 197–210. ISBN: 978-3-031-31209-0 (pages 34, 35).

[31] Intel. *oneAPI*. 2023. URL: https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html (pages 34, 36–38).

References VIII

[32] Intel. *Intel Application Migration Tool for OpenACC to OpenMP API*. 2023. URL: `https://github.com/intel/intel-application-migration-tool-for-openacc-to-openmp` (page 36).

[33] Intel. *Data Parallel Control*. 2023. URL: `https://github.com/IntelPython/dpctl` (page 39).

[34] Intel. *Data-parallel Extension to Numba*. 2023. URL: `https://github.com/IntelPython/numba-dpex` (page 39).

[35] Intel. *Data Parallel Extension for Numpy*. 2023. URL: `https://github.com/IntelPython/dpnp` (page 39).