



AdaptiveCpp (formerly known as hipSYCL / Open SYCL)

(Note: This project is currently in progress of changing its name to AdaptiveCpp due to external legal pressure. Documentation and code may still use the older name hipSYCL / Open SYCL)

AdaptiveCpp is the independent, community-driven modern platform for C++-based heterogeneous programming models targeting CPUs and GPUs from all major vendors. AdaptiveCpp lets applications adapt themselves to all the hardware found in the system. This includes use cases where a single binary needs to be able to target all supported hardware, or utilize hardware from different vendors simultaneously.

It currently supports the following programming models: 1. **SYCL**: At its core is a highly competitive and flexible SYCL implementation that supports many compilation flows. 2. **C++ standard parallelism**: Additionally, AdaptiveCpp features experimental support for offloading C++ algorithms from the parallel STL. See here for details on which algorithms can be offloaded. **AdaptiveCpp is currently the only solution capable of demonstrating C++ standard parallelism performance across NVIDIA, AMD and Intel GPUs, and in most cases outperforms vendor compilers.**

AdaptiveCpp supports CPUs (including x86, arm and other LLVM-supported architectures) as well as GPUs from Intel, NVIDIA, and AMD. This includes the ability to generate a single binary that can offload to all supported devices.

AdaptiveCpp's compilation flows include 1. A powerful, generic LLVM JIT compiler. This is AdaptiveCpp's default, most portable and usually most performant compilation flow. It is also the **world's only SYCL compiler that only needs to parse the source code a single time** across both host and device compilation. 2. Compilation flows focused on providing interoperability at source code level with vendor programming models (including e.g. the ability to mix-and-match CUDA and SYCL in the same source file) 3. Library-only compilation flows focused on deployment simplicity. These compilation flows allow utilizing third-party compilers, with AdaptiveCpp merely acting as a library.

A full list of its compilation capabilities can be found [here](#).

Because a program compiled with AdaptiveCpp appears just like any other program written in vendor-supported programming models (like CUDA or HIP) to vendor-provided software, vendor tools such as profilers or debuggers also work with AdaptiveCpp.

An illustration on how the project fits into the SYCL ecosystem can be found (here).

Performance & benchmarking

See the AdaptiveCpp performance guide.

Installing and using AdaptiveCpp

- Building & Installing

In order to compile software with AdaptiveCpp, use `acpp`. `acpp` can be used like a regular compiler, i.e. you can use `acpp -o test test.cpp` to compile your application called `test.cpp` with AdaptiveCpp.

`acpp` accepts both command line arguments and environment variables to configure its behavior (e.g., to select the target to compile for). See `acpp --help` for a comprehensive list of options.

For details and instructions on using AdaptiveCpp in CMake projects, please see the documentation on using AdaptiveCpp.

About the project

Development of AdaptiveCpp is currently primarily led by Heidelberg University, with contributions from a growing community. We see AdaptiveCpp as a community-driven project, in contrast to the many vendor-driven heterogeneous compiler efforts. AdaptiveCpp not only serves as a research platform, but is also a solution used in production on machines of all scales, including some of the most powerful supercomputers.

Getting in touch

Join us on Discord! Alternatively, open a discussion or issue in this repository.

Contributing to AdaptiveCpp

We encourage contributions and are looking forward to your pull request! Please have a look at CONTRIBUTING.md. If you need any guidance, please just open an issue and we will get back to you shortly.

If your institution or organization is considering to support the AdaptiveCpp development in some official capacity, we are always happy to discuss collaborations and to broaden the developer community. Please do reach out :-)

If you are a student at Heidelberg University and wish to work on AdaptiveCpp, please get in touch with us. There are various options possible and we are happy to include you in the project :-)

Citing AdaptiveCpp

AdaptiveCpp is a production platform for heterogeneous computing, but also a research project. As such, if you use AdaptiveCpp in your research, we kindly request that you cite one of the following publications, depending on your focus:

- A general overview, SYCL 2020, performance and the relationship with oneAPI: Aksel Alpay, Bálint Soproni, Holger Wünsche, and Vincent Heuveline. 2022. *Exploring the possibility of a hipSYCL-based implementation of oneAPI*. In *International Workshop on OpenCL (IWOCCL'22)*. Association for Computing Machinery, New York, NY, USA, Article 10, 1–12. <https://doi.org/10.1145/3529538.3530005>
- The generic single-pass compiler: Aksel Alpay and Vincent Heuveline. 2023. *One Pass to Bind Them: The First Single-Pass SYCL Compiler with Unified Code Representation Across Backends*. In *Proceedings of the 2023 International Workshop on OpenCL (IWOCCL '23)*. Association for Computing Machinery, New York, NY, USA, Article 7, 1–12. <https://doi.org/10.1145/3585341.3585351>
- Our CPU compiler: Joachim Meyer, Aksel Alpay, Sebastian Hack, Holger Fröning, and Vincent Heuveline. 2023. *Implementation Techniques for SPMD Kernels on CPUs*. In *Proceedings of the 2023 International Workshop on OpenCL (IWOCCL '23)*. Association for Computing Machinery, New York, NY, USA, Article 1, 1–12. <https://doi.org/10.1145/3585341.3585342>
- The original talk and the idea of implementing SYCL on non-OpenCL backends: Aksel Alpay and Vincent Heuveline. 2020. *SYCL beyond OpenCL: The architecture, current state and future direction of hipSYCL*. In *Proceedings of the International Workshop on OpenCL (IWOCCL '20)*. Association for Computing Machinery, New York, NY, USA, Article 8, 1. DOI:<https://doi.org/10.1145/3388333.3388658>

(The latter is a talk and available online. Note that some of the content in this talk is outdated by now)

Acknowledgements

We gratefully acknowledge contributions from the community.

Documentation

- AdaptiveCpp design and architecture
- AdaptiveCpp runtime specification
- AdaptiveCpp compilation model
- How to use raw HIP/CUDA inside AdaptiveCpp code to create optimized code paths
- A simple SYCL example code for testing purposes can be found [here](#).
- SYCL Extensions implemented in AdaptiveCpp
- Macros used by AdaptiveCpp
- Environment variables supported by AdaptiveCpp