June 13, 2023

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

hipSYCL's quest for universal SYCL binaries

One binary from NVIDIA and AMD to Intel Data Center GPU Max Series

Aksel Alpay
Heidelberg University

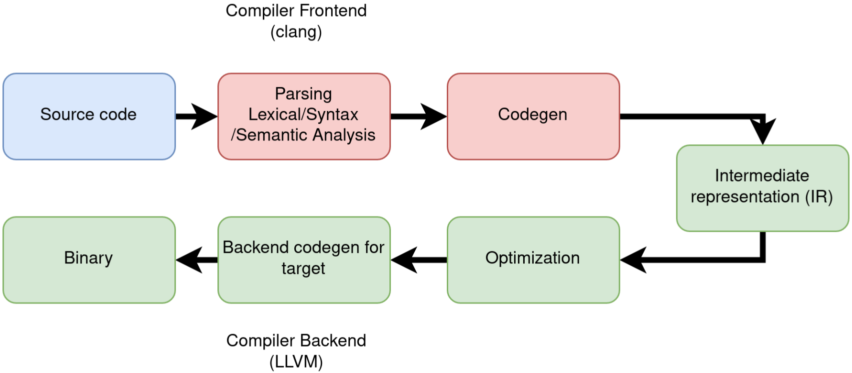
Universal binaries with SYCL?

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- What if SYCL binaries need to be deployed and run on systems with unknown hardware configuration?
  - Usually not important for HPC
  - …but central e.g. for vendors of software who want to distribute binaries to their end users!
  - In this case, a single ("universal") binary needs to be able to run on whatever hardware is available on the target system.
- How is this adressed in current SYCL implementations?

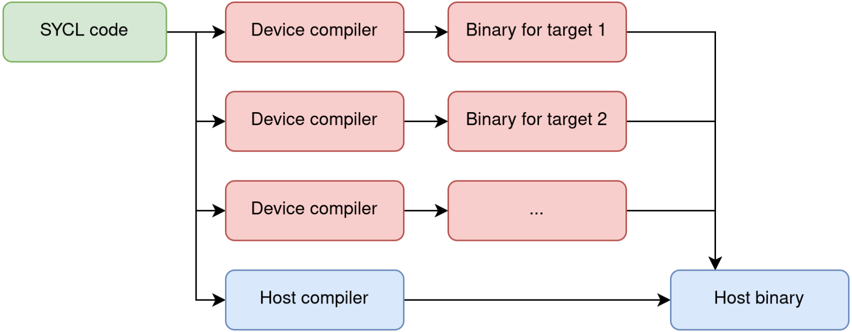
Review of current SYCL compilation models

Sketch of clang/LLVM architecture



How can this be extended to heterogeneous architectures, and create binaries that run not only on CPU, but also on e.g. GPU?

SMCP (Single-source, multiple compiler passes)



- Each compiler invocation parses code again, and emits a binary in a backend-specific format
- All SYCL implementations, and almost all heterogeneous compilers (e.g. nvcc) use this model

SMCP trouble

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Parsing modern C++ code is expensive
- As we add more backends and support for more hardware, compile times do not scale well!
- Some backends (ROCm) do not have an intermediate format – every GPU needs to be targeted explicitly
- Creating a binary using SMCP that runs on all hardware supported by hipSYCL requires roughly 40 compilation passes…(similarly for e.g. DPC++)
- **Very difficult (impractical?) to create a SYCL binary that "just runs" on any GPU**

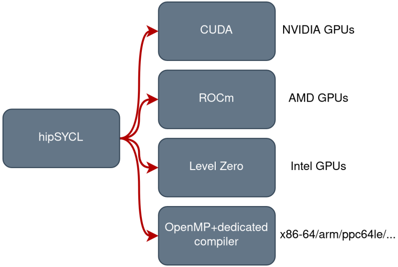
Portable interchange formats (e.g. SPIR-V) to the rescue?

- Not universally supported
- Still want interop with backend-specific libraries, and hence need to rely on vendor-specific SYCL backends (e.g. CUDA/HIP) which rely on their own device binary format

hipSYCL's new compiler: Designed for universal binaries

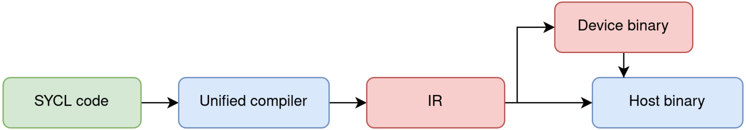
A brand-new compiler for hipSYCL

- **First SYCL compiler with unified code representation across backends** (Single device binary embedded in the application that is shared across CUDA, ROCm, Level Zero backends)
- **First single-pass SYCL compiler** (source is only parsed once, unified compiler that generates both host and device code)



All developments are publicly available as part of the hipSYCL SYCL implementation.

SSCP (Single-source, single compiler pass)



- Code is only parsed once – allows for faster compile times
- Not a new idea, but rarely used: Only major production compiler: NVIDIA nvc++.
- Until now, this has not been done in a stand-alone SYCL compiler.

Embedded device binary follows a generic code representation!

- Analyses/Optimizations such as kernel fusion can be implemented in a backend-independent way – accelerated development!
- Clear path to extend hipSYCL support to new hardware

How it works

Alpay A., Heuveline V. 2023. One Pass to Bind Them: The First Single-Pass SYCL Compiler with Unified Code Representation Across Backends. IWOCL'23. https://doi.org/10.1145/3585341.3585351

Details in the paper!

General overview

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Stage 1

- Typically happens at compile time (target device not yet known)
- During regular host compilation, identify & extract kernels IR from host IR
- Make sure LLVM IR does not contain target-specific hints/builtin calls/…
- Embed LLVM IR bitcode for kernels in host application

Stage 2

- Typically happens at runtime (target device known)
- Take generic LLVM IR and compile to backend-specific format for device we want to run on (e.g. NVIDIA PTX)
- Optimize

Code specialization via control flow

We need to be able to have different code paths for host/device or different device targets

- Users may want to create dedicated target-optimized code paths
- Implementation needs to figure out how to map e.g. builtins.
- SYCL specification: **If** you use SMCP, you must define `__SYCL_DEVICE_ONLY__` macro in the device pass

Macros cannot be used with SSCP for this, because code is only parsed once!

```cpp
double sin(double x) {
  if(__hipsycl_sscp_is_device) {
    return __device_sin(x);
  } else {
    return std::sin(x);
  }
}
```

- `__hipsycl_sscp_is_device` is an **IR constant**
- There are IR constants for stage 1 and stage 2 compilation. Stage 2 IR constants can also be added by the user.
- Since IR constants are seen as constants for the optimizer, additional dead code elimination passes removes unneded branches

No branches will be in compiled code anymore!

IR constants: Global variable, which is not a C++ constant, but will be turned into a constant by the compiler in LLVM IR, and initialized with a value that is not known yet when code is parsed.

Current support

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- NVIDIA, Intel, AMD GPUs. CPUs are planned.
- Not all SYCL functionality is supported yet
  - Group algorithms (WIP)
  - SYCL 2020 reductions
  - Some extensions

Results

Hardware:

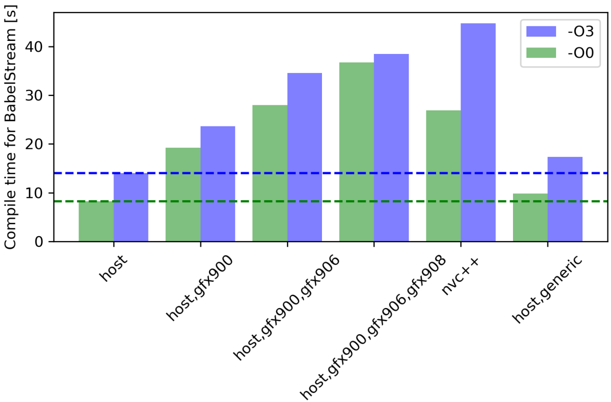
- System 1: AMD Ryzen 7 4750U APU, 32GB RAM, Arch Linux
- System 2: Intel Core i7 8550U CPU with iGPU, 16GB RAM, NVIDIA GeForce MX150 GPU, Ubuntu 22.04
- System 3: Intel Core i7 8700 CPU, 64GB RAM, AMD Radeon Pro VII GPU, Ubuntu 20.04

Benchmarks:

- BabelStream (memory benchmarks [Deakin et al. (2016)])
- CloverLeaf (2D hydrodynamics mini-app [Deakin et al. (2020)])
- miniBUDE (molecular docking mini-app [Poenaru et al. (2021)])
- RSBench/XSBench (monte-carlo neutron transport mini-apps [Tramm et al. (2014)])

Compile Time: BabelStream

`host,generic` vs other hipSYCL compilation flows



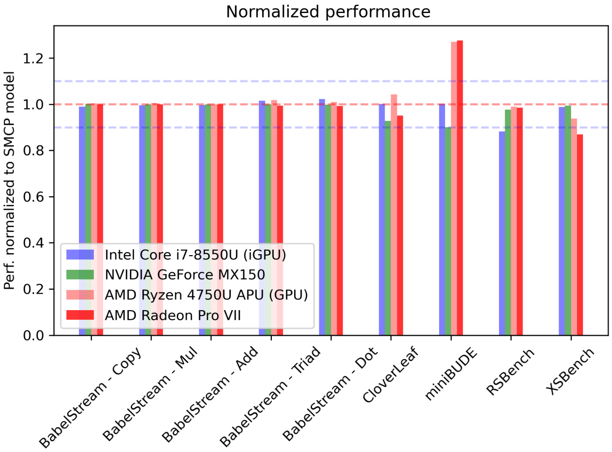
- Generic SSCP compiler only $\approx$ 20% slower than a regular clang host compilation
- This overhead is due to clang OpenMP frontend, disable using `-fno-openmp`
- Note: the other compilation flows are slower, while targeting less hardware!

But have you just moved compile times to runtime?

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Previous SYCL compilers might do more work then necessary, since all device images for all backends need to be created ahead-of-time, but not all might be needed on the machine of the user
- Stage 2 compilation does not require parsing the code, so parsing costs have been removed
- Note that even in SYCL implementations today there is already runtime compilation taking place!
  - Drivers need to translate backend-specific IRs like PTX, SPIR-V to machine code
  - **SYCL applications need to already expect runtime compilation overheads and deal with this today!**
  - Our additional runtime compilation logic can be expected to add around $0.2\times$ to $1\times$ of the existing runtime compilation overheads to the first kernel invocation
  - Existing strategies in SYCL apps to deal with overheads at the first kernel invocation will likely still work.

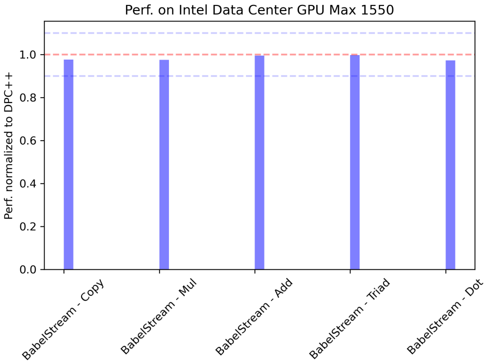
Performance



- Performance within -13% to +27%
- Intel results are normalized to DPC++ perf, since the old hipSYCL SMCP SPIR-V compiler was too immature

Note: Development focus so far was functionality, not performance!

Also works on Intel Data Center GPU Max Series devices!



- Only short amount of time available during SYCL hackathon @ IWOCL
- Performance within few percents
- Unclear if differences are even due to the generated code

Not a single line of code had to be changed in hipSYCL – the power of open standards! (Level Zero/SPIR-V)

Conclusion

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- hipSYCL's generic single-pass compiler delivers **significantly lower compile times** (especially when targeting multiple backends/devices) and **instant binary portability**, while **retaining performance**.
- **Single binary** that can adapt its embedded kernel code based on the hardware it finds on the system
- That one binary can then be executed on **NVIDIA, AMD and Intel GPUs** - including **Intel Data Center GPU Max Series devices**!
- **Robust platform to extend hipSYCL** to new hardware, or implement features like profile guided optimizations, kernel fusion, ...in a backend-independent manner
- Publicly available & works today even with complex applications like CloverLeaf
- Super easy to use: Just compile with `--hipsycl-targets=generic`

What's left to do?

- For users: Install hipSYCL and try it out today! Give feedback!
- For academics and compiler engineers: Read the paper! Alpay A., Heuveline V. 2023. One Pass to Bind Them: The First Single-Pass SYCL Compiler with Unified Code Representation Across Backends. IWOCL'23. https://doi.org/10.1145/3585341.3585351
- Reach out!
  - `aksel.alpay@uni-heidelberg.de`
  - `@illuhad` on Twitter & Reddit, `@illuhad@mastodon.world` on Mastodon.

- **hipSYCL is now the first generic single-pass SYCL implementation! One compiler pass, one binary, all the devices.**