April 26, 2021

UNIVERSITÄTS-
RECHENZENTRUM

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

SYCL 2020 in hipSYCL

DPC++ features on AMD GPUs, NVIDIA GPUs and CPUs

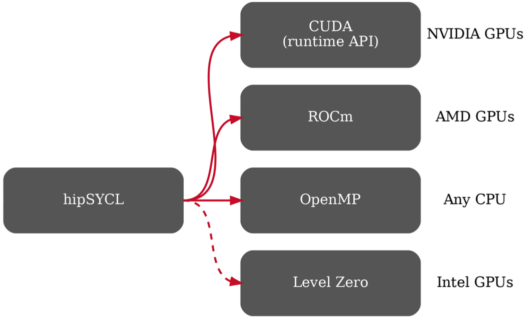
Aksel Alpay
Heidelberg University

Introduction to hipSYCL

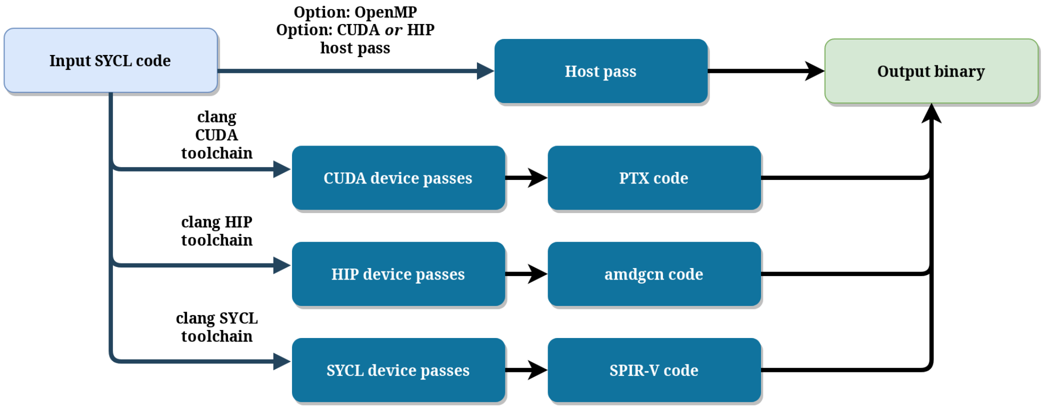
hipSYCL

A generic, multi-backend SYCL implementation with emphasis on aggregating existing toolchains.

- Source-compatible with vendor-specific programming models
- Unique extensions, e.g. full buffer-USM interoperability

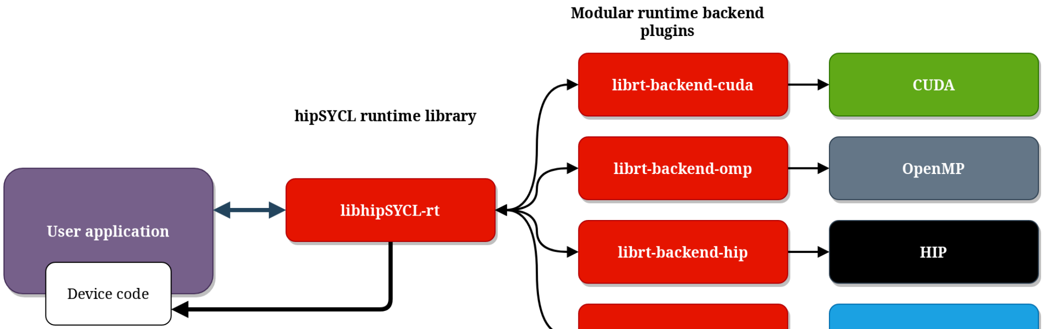


hipSYCL: Multiple toolchains in one.



- `syclcc -O3 --hipsycl-targets="omp;cuda:sm_70;hip:gfx906" test.cpp`
- CMake integration available: `find_package(hipSYCL)`, `add_sycl_to_target()`

hipSYCL runtime architecture



SYCL 2020: Simple things are simple now

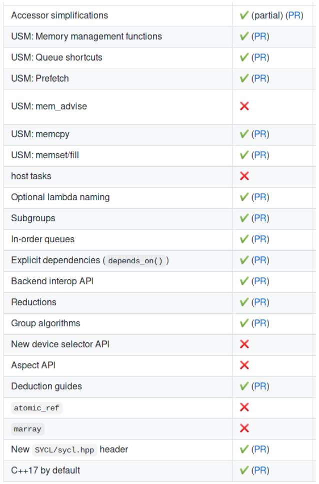
```
using namespace access;
queue q;
{
 buffer<int> a{input_a, size}
 buffer<int> b{input_b, size};
 buffer<int> c{output, size};
 q.submit([&](handler& cgh){
  auto aa=a.get_access<mode::read>();
  auto ab=b.get_access<mode::read>();
  auto ac=c.get_access<mode::write>();
  cgh.parallel_for<class name>(size
      ,[=](sycl::idx<1> i){
   ac[i] = aa[i] + ab[i];
  });
 });
}
```

```
queue q;
int* a=malloc_shared<int>(size);
int* b=malloc_shared<int>(size);
int* c=malloc_shared<int>(size);
//TODO: Fill input a,b
q.parallel_for(size,[=](id<1> i){
 c[i] = a[i] + b[i];
}).wait();
```

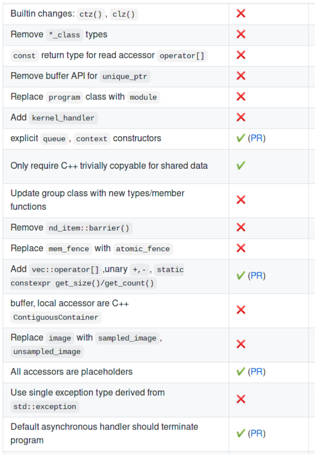
SYCL 2020 in hipSYCL

UNIVERSITÄTS-RECHENZENTRUM

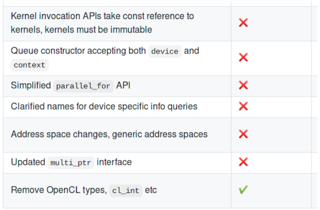
UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386



Accessor simplifications	✓ (partial) (PR)
USM: Memory management functions	✓ (PR)
USM: Queue shortcuts	✓ (PR)
USM: Prefetch	✓ (PR)
USM: mem_advise	✗
USM: memcpy	✓ (PR)
USM: memset/fill	✓ (PR)
host tasks	✗
Optional lambda naming	✓ (PR)
Subgroups	✓ (PR)
In-order queues	✓ (PR)
Explicit dependencies (`depends_on()`)	✓ (PR)
Backend interop API	✓ (PR)
Reductions	✓ (PR)
Group algorithms	✓ (PR)
New device selector API	✗
Aspect API	✗
Deduction guides	✓ (PR)
`atomic_ref`	✗
`marray`	✗
New `SYCL/sycl.hpp` header	✓ (PR)
C++17 by default	✓ (PR)



Builtin changes: `ctz()` , `clz()`	✗
Remove `*_class` types	✗
`const` return type for read accessor `operator[]`	✗
Remove buffer API for `unique_ptr`	✗
Replace `program` class with `module`	✗
Add `kernel_handler`	✗
explicit `queue` , `context` constructors	✓ (PR)
Only require C++ trivially copyable for shared data	✓
Update group class with new types/member functions	✗
Remove `nd_item::barrier()`	✗
Replace `mem_fence` with `atomic_fence`	✗
Add `vec::operator[]` ,unary `+,-` , `static constexpr get_size()/get_count()`	✓ (PR)
buffer, local accessor are C++ `ContiguousContainer`	✗
Replace `image` with `sampled_image` , `unsampled_image`	✗
All accessors are placeholders	✓ (PR)
Use single exception type derived from `std::exception`	✗
Default asynchronous handler should terminate program	✓ (PR)



Kernel invocation APIs take const reference to kernels, kernels must be immutable	✗
Queue constructor accepting both `device` and `context`	✗
Simplified `parallel_for` API	✗
Clarified names for device specific info queries	✗
Address space changes, generic address spaces	✗
Updated `multi_ptr` interface	✗
Remove OpenCL types, `cl_int` etc	✓

https://github.com/hipSYCL/featuresupport

Key DPC++/SYCL 2020 features implemented

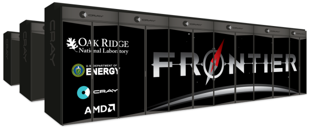
Our work as oneAPI Center of Excellence:

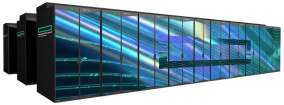
- Unified Shared Memory (USM)
- Optional Lambda Naming
- Subgroups
- Parallel algorithms for groups and subgroups
- Parallel reductions
- Queue shortcuts
- Explicit task graphs
- ...

hipSYCL support increases adoption and portability of SYCL 2020 features (e.g. AMD GPUs)

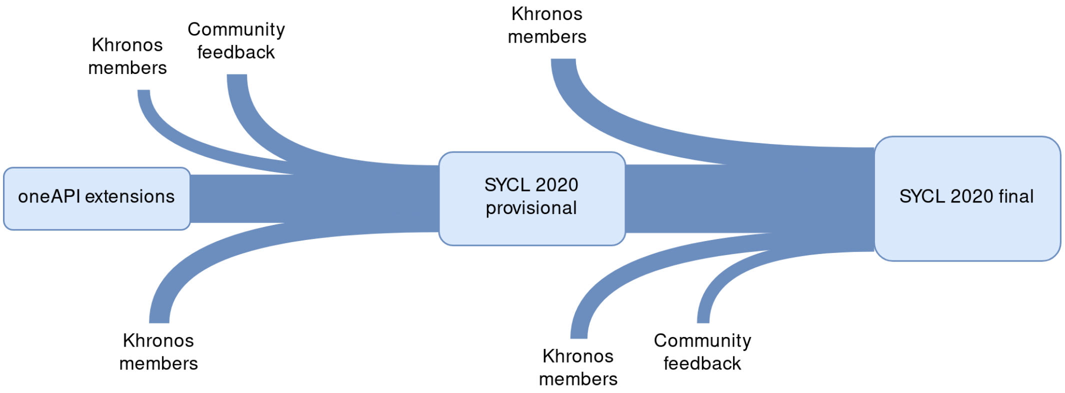
Frontier, El Capitan, LUMI

Bring SYCL 2020 to upcoming supercomputers based on AMD GPUs



U.S. DEPARTMENT OF ENERGY
NNSA
Lawrence Livermore National Laboratory
CRAY
EL CAPITAN


From DPC++ extensions to SYCL 2020



- SYCL 2020 interfaces in current implementations and client code may vary
- hipSYCL interfaces started with SYCL 2020 provisional, now moving towards SYCL 2020 final

Unified Shared Memory in hipSYCL

- Device-accessible host memory
- Explicit USM
- Shared allocations (on-demand migration)
- Performance hints (prefetch/memadvise)

```
sycl::queue q;
int* ptr =
 sycl::malloc_shared<int>(size,q)
q.parallel_for(size,
  [=](sycl::id<1> idx){
 const int i = idx.get(0);
 ptr[i] = i;
});
```

CUDA
`cudaMallocHost`
`cudaMalloc`
`cudaMallocManaged`
`cudaMemPrefetchAsync`
`cudaMemAdvise`

HIP
`hipHostMalloc`
`hipMalloc`
`hipMallocManaged`
`hipMemPrefetchAsync`
`hipMemAdvise`

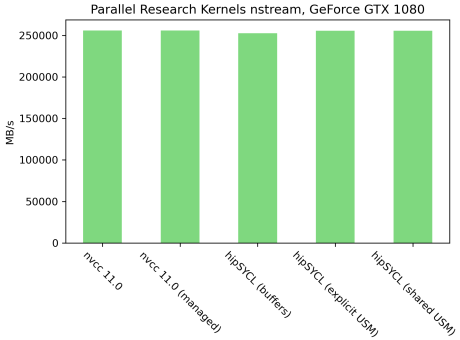
CPU
(regular host allocations)

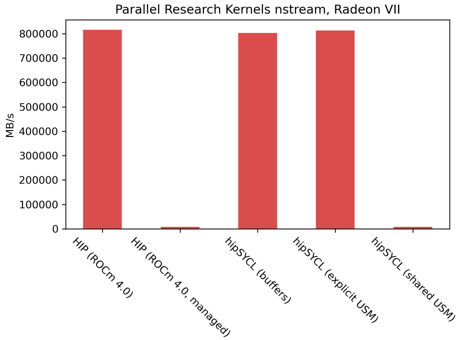
hipSYCL USM performance

UNIVERSITÄTS-RECHENZENTRUM

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Parallel research kernels benchmarks[1]





ROCm does not yet fully support allocations with on-demand page migration.

[1] https://github.com/ParRes/Kernels

Subgroups

UNIVERSITÄTS-RECHENZENTRUM

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Expose hardware below work group granularity
- SIMD units
- Useful for optimization

```
sycl::nd_item<1> idx = ...;
auto sgrp = idx.get_sub_group();
```

CUDA
Mapped to CUDA warps

HIP
Mapped to AMD wave-fronts

CPU
Individual subgroup for each work item

- Difficult for library-only CPU backends
- Vectorization controlled by OpenMP compiler

Group algorithms in hipSYCL

- `any_of`, `none_of`, `all_of`
- Reductions, scans
- broadcast, barrier
- At work group and subgroup level
- Collective and Iterator-based variants

```
int myval = ...;
int sum=sycl::reduce_over_group(
 group, myval, sycl::plus<int>{})
```

CUDA

- Subgroup intrinsics, warp shuffles
- Optimized local memory usage

HIP

- Subgroup intrinsics, warp shuffles
- Optimized local memory usage

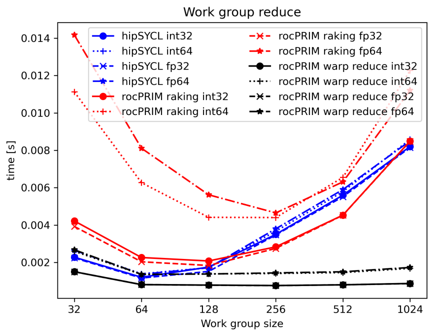
CPU

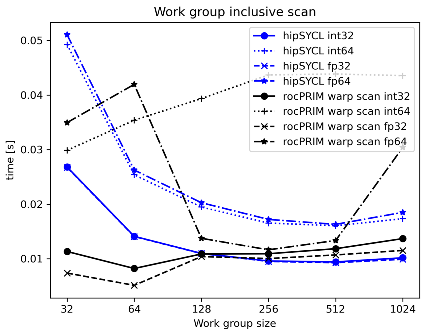
- Sequential with OpenMP vectorization
- Bound by synchronization

Group reduce and scan on AMD Radeon VII

UNIVERSITÄTS-RECHENZENTRUM

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386





- Competitive performance compared to rocPRIM
- We are handicapped: Group size not known at compile time

Wünsche, H. (2021): "SYCL 2020 work group parallel primitives [...] in hipSYCL". Bachelor Thesis. Heidelberg U.

Parallel reductions

- Variadic scalar reductions
- basic, `nd_range`, hierarchical and scoped parallelism supported
- No multi-dimensional reductions yet
- Huge optimization space!

```
q.parallel_for(range{size},
 reduction(output,
           sycl::plus<int>{}),
 [=](sycl::id<1> idx,
     auto& reducer){
  int myvalue = ...;
  reducer += myvalue;
});
```

CUDA

- Work group reductions
- Multiple kernel launches

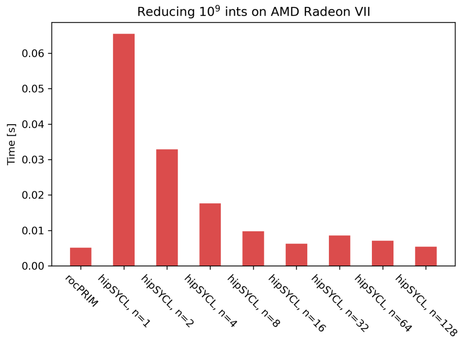
HIP

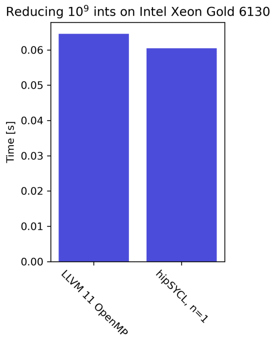
- Work group reductions
- Multiple kernel launches

CPU

- Per-thread reductions to cache-line aligned private storage
- Cross-thread final reduction

Reduction performance





- hipSYCL reduction performance can compete with vendor-optimized libraries
- SYCL model is very flexible and allows for more user control – and user error!
- **For pure reductions, we still need optimized SYCL libraries!**

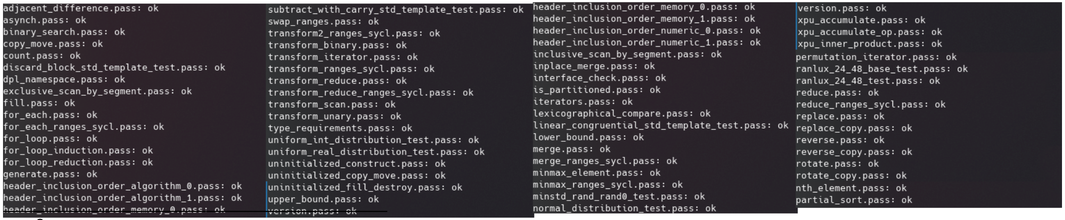
oneAPI libraries

UNIVERSITÄTS-RECHENZENTRUM

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

The language is only half the way for oneAPI portability! We need the libraries.

- oneMKL: hipSYCL (and rocBLAS) support currently WIP
- oneDPL: Most unit tests already run with hipSYCL[2]



```
adjacent_difference.pass: ok
asynch.pass: ok
binary_search.pass: ok
copy_move.pass: ok
count.pass: ok
discard_block_std_template_test.pass: ok
dpl_namespace.pass: ok
exclusive_scan_by_segment.pass: ok
fill.pass: ok
for_each.pass: ok
for_each_ranges_sycl.pass: ok
for_loop.pass: ok
for_loop_induction.pass: ok
for_loop_reduction.pass: ok
generate.pass: ok
header_inclusion_order_algorithm_0.pass: ok
header_inclusion_order_algorithm_1.pass: ok
header_inclusion_order_memory_0.pass: ok
```

```
subtract_with_carry_std_template_test.pass: ok
swap_ranges.pass: ok
transform2_ranges_sycl.pass: ok
transform_binary.pass: ok
transform_iterator.pass: ok
transform_ranges_sycl.pass: ok
transform_reduce.pass: ok
transform_reduce_ranges_sycl.pass: ok
transform_scan.pass: ok
transform_unary.pass: ok
type_requirements.pass: ok
uniform_int_distribution_test.pass: ok
uniform_real_distribution_test.pass: ok
uninitialized_construct.pass: ok
uninitialized_copy_move.pass: ok
uninitialized_fill_destroy.pass: ok
upper_bound.pass: ok
version.pass: ok
```

```
header_inclusion_order_memory_0.pass: ok
header_inclusion_order_memory_1.pass: ok
header_inclusion_order_numeric_0.pass: ok
header_inclusion_order_numeric_1.pass: ok
inclusive_scan_by_segment.pass: ok
inplace_merge.pass: ok
interface_check.pass: ok
is_partitioned.pass: ok
iterators.pass: ok
lexicographical_compare.pass: ok
linear_congruential_std_template_test.pass: ok
lower_bound.pass: ok
merge.pass: ok
merge_ranges_sycl.pass: ok
minmax_element.pass: ok
minmax_ranges_sycl.pass: ok
minstd_rand_rand0_test.pass: ok
normal_distribution_test.pass: ok
```

```
version.pass: ok
xpu_accumulate.pass: ok
xpu_accumulate_op.pass: ok
xpu_inner_product.pass: ok
permutation_iterator.pass: ok
ranlux_24_48_base_test.pass: ok
ranlux_24_48_test.pass: ok
reduce.pass: ok
reduce_ranges_sycl.pass: ok
replace.pass: ok
replace_copy.pass: ok
reverse.pass: ok
reverse_copy.pass: ok
rotate.pass: ok
rotate_copy.pass: ok
nth_element.pass: ok
partial_sort.pass: ok
```

[2] https://github.com/hipSYCL/oneDPL

Case study: oneDPL

What was necessary to port oneDPL to hipSYCL?

- Build system
- SYCL compiler detection/macros
- pre-SYCL 2020 final APIs:
  - `ONEAPI` namespace for reductions/group algorithms
  - `noinit` vs `no_init` property
- DPC++ implementation details
  - `mode_tag_t`
- Optimize work group size selection for hipSYCL

- **No functional changes**
- **Single file with some compatibility aliases**

Conclusion

hipSYCL

- Rapidly moving towards SYCL 2020
- Key features supported - high performance implementations for all backends.

It is possible to write standard SYCL 2020/DPC++ code, be portable without sacrificing performance w.r.t vendor-optimized libraries!

- Make SYCL 2020 ubiquitous - let DPC++/SYCL 2020 code run on AMD GPUs, NVIDIA GPUs, any CPU
- Ongoing work: oneMKL, oneDPL, other SYCL 2020 features, more optimizations...
- Open source: `https://github.com/illuhad/hipSYCL`
- Get in touch: `aksel.alpay@uni-heidelberg.de`