OAK RIDGE National Laboratory

Experiences with Kokkos' SYCL Backend using AMD GPUs

Daniel Arndt, Damien Lebrun-Grandié, Christian Trott

April 10, 2025

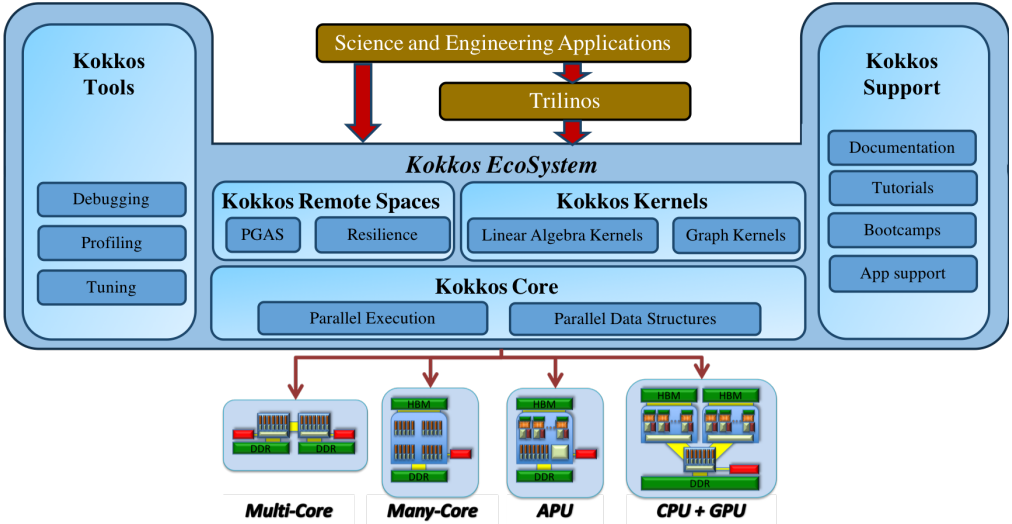
ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

What is Kokkos?

- A C++ Programming Model for Performance Portability
  - Template library on top CUDA, HIP, OpenMP, SYCL, ...
  - Aligns with developments in the C++ standard, e.g., `mdspan`, `atomic_ref`
- Expanding solution for common needs of modern science and engineering codes
  - Math libraries based on Kokkos
  - Tools for debugging, profiling and tuning
  - Interoperability with Fortran and Python
- Open Source project with a growing community
  - Maintained and developed at `https://github.com/kokkos`
  - Hundreds of users at many large institutions
- Philosophy: Support vendor-provided toolchain

Kokkos EcoSystem



Kokkos - Applications

More than 50% of C++ Codes in the Exascale Computing Project use Kokkos. Example of Applications:

- Trilinos (Linear Algebra)
- PETSc (Linear Algebra)
- deal.II (Finite Element Library)
- LAMMPS (Molecular Dynamics)
- XGC (Fusion Reactor Simulation)
- ArborX (Geometry Search)
- Uintah (Chemical Reactions)
- VTK-m (Visualization)
- ...

Kokkos Core Functionalities, Mapping to SYCL

Constructs

- `parallel_for` ⟶ `sycl::parallel_for`
- `parallel_reduce` ⟶ `sycl::parallel_for`
- `parallel_scan` ⟶ `sycl::parallel_for`

Policies

- `RangePolicy` ⟶ `sycl::range`
- `MDRangePolicy` ⟶ `sycl::nd_range`
- `TeamPolicy` ⟶ `sycl::nd_range`

Memory

- `View` ⟶ `sycl::malloc/sycl::free`

Used Extensions

- `sycl::ext::oneapi::experimental::this_nd_item`
- `sycl::ext::oneapi::experimental::printf`
- `sycl::ext::oneapi::experimental::device_global`
- `sycl::ext::oneapi::experimental::properties`
- `sycl::ext::oneapi::experimental::this_sub_group`
- `sycl::ext::oneapi::group_ballot`
- `sycl::ext::oneapi::sub_group_mask`
- `sycl::ext::oneapi::experimental::bfloat16`
- `sycl::ext::oneapi::group_local_memory_for_overwrite`
- `sycl::ext::oneapi::experimental::command_graph`

IWOCL 2024

Experiences with Implementing Kokkos' SYCL Backend

- Intel GPUs and Nvidia GPUs work
- Why USM not `sycl_buffers`
- `is_device_copyable`, `printf` crucial
- Lacking abstraction for hierarchical parallelism in SYCL

What about AMD GPUs?

oneAPI on Frontier

There is a module for oneAPI 2024.2.0 with ROCm 5.4.3 but we get many ICEs in

- Kokkos_CoreUnitTest_SYCL1A.dir/sycl/TestSYCL_DeepCopyAlignment.cpp
- Kokkos_CoreUnitTest_SYCL1B.dir/sycl/TestSYCL_MDRangeReduce.cpp
- Kokkos_CoreUnitTest_SYCL3.dir/sycl/TestSYCLSharedUSM_ViewAPI_e.cpp
- Kokkos_AlgorithmsUnitTest_StdSet_Team_B
- Kokkos_AlgorithmsUnitTest_StdSet_Team_C
- Kokkos_AlgorithmsUnitTest_StdSet_C

oneAPI on Frontier

Tests that are compiling:

- Kokkos_CoreUnitTest_SYCL_ViewSupport (Subprocess aborted)
- Kokkos_CoreUnitTest_Serial1 (Failed)
- Kokkos_CoreUnitTest_SYCL1A (Not Run)
- Kokkos_CoreUnitTest_SYCL1B (Not Run)
- Kokkos_CoreUnitTest_SYCL2A (Timeout)
- Kokkos_CoreUnitTest_SYCL3 (Not Run)
- Kokkos_CoreUnitTest_SYCLInterOpInit (SEGFAULT)
- Kokkos_CoreUnitTest_SYCLInterOpStreams (SEGFAULT)
- Kokkos_CoreUnitTest_SYCLInterOpGraph (Failed)
- Kokkos_CoreUnitTest_Default (Failed)
- Kokkos_ContainersUnitTest_SYCL (Failed)
- Kokkos_UnitTest_Sort (Failed)
- Kokkos_UnitTest_Random (Failed)
- Kokkos_AlgorithmsUnitTest_StdSet_C (Not Run)
- Kokkos_AlgorithmsUnitTest_StdSet_D (Failed)
- Kokkos_AlgorithmsUnitTest_StdSet_E (Failed)
- Kokkos_AlgorithmsUnitTest_StdSet_Team_B (Not Run)
- Kokkos_AlgorithmsUnitTest_StdSet_Team_C (Not Run)
- Kokkos_AlgorithmsUnitTest_StdSet_Team_I (Failed)
- Kokkos_AlgorithmsUnitTest_StdSet_Team_L (Failed)

oneAPI on Frontier

```
/autofs/nccs-svm1_sw/frontier/ums/ums015/oneapi/2024.2.0/compiler/2024.2/bin/compiler/../
  include/sycl/ext/oneapi/experimental/builtins.hpp:84:36: error: SYCL kernel cannot call
  a variadic function
   84 |   return ::printf(__format, args...);
      |                                    ^
/ccs/home/darndt/kokkos/core/src/Kokkos_Printf.hpp:43:38: note: called by
  'printf<const char *>'
   43 |     sycl::ext::oneapi::experimental::printf(format, args...);
      |                                      ^
```

oneAPI on Frontier

Newer toolchains (DPC++ 2025.0.0, ROCm 6.1.3) much better

- we still can't use 'printf' resulting in segmentation faults.
- problems in shared memory reductions ⇒use shuffle-based reductions with reduced functionality (no array reductions)
- device global variables cause segmentation faults
- `sycl::memory_order::seq_cst` not available for `sycl::atomic_ref`
- Failing tests
  - sycl_graph.submit_six (also failing on NVIDIA GPUs)
  - std_algorithms_unique_team_test.test_default_predicate
  - std_algorithms_partition_copy_team_test.all_true
  - std_algorithms_partition_copy_team_test.random

oneAPI on Frontier - shared memory problems I

Writes here...

```
// Perform the actual workgroup reduction in each subgroup separately.
auto sg            = item.get_sub_group();
auto* result = &local_mem[local_id * value_count];
const int id_in_sg = sg.get_local_id()[0];
const auto local_range = std::min(sg.get_local_range()[0], max_size);
const auto upper_stride_bound = (local_range - id_in_sg, max_size - local_id);
for (unsigned int stride = 1; stride < local_range; stride <<= 1) {
  if (stride < upper_stride_bound)
    final_reducer.join(result, &local_mem[(local_id + stride) * value_count]);
  sycl::group_barrier(sg);
}
sycl::group_barrier(item.get_group());
```

oneAPI on Frontier - shared memory problems II

...aren't observed here

```
// Do the final reduction only using the first subgroup.
if (sg.get_group_id()[0] == 0) {
  const unsigned int n_subgroups = sg.get_group_range()[0];
  const int max_subgroup_size    = sg.get_max_local_range()[0];
  auto* result_ = &local_mem[id_in_sg * max_subgroup_size * value_count];
  for (unsigned int offset = local_range;
       offset < std::min(n_subgroups, max_size); offset += local_range)
    if (id_in_sg + offset < n_subgroups)
      final_reducer.join(result_,
          &local_mem[(id_in_sg + offset) * max_subgroup_size * value_count]);
  sycl::group_barrier(sg);
  [...]
}
```

oneAPI on Frontier - launch failures

```
11: [ RUN      ] sycl.uvm
11: <HIP>[ERROR]:
11: UR HIP ERROR:
11:     Value:           209
11:     Name:            hipErrorNoBinaryForGpu
11:     Description:     no kernel image is available for execution on the device
11:     Function:        buildProgram
11:     Source Location: [...]
11:
11: unknown file: Failure
11: C++ exception with description "The program was built for 1 devices
11: Build program log for 'AMD Instinct MI210': ^G" thrown in the test body.
```

oneAPI on Frontier

Other tests with the same problem

- sycl.raw_sycl_interop
- sycl.raw_sycl_interop_context_1
- sycl.raw_sycl_interop_context_2
- kokkosp.test_kernel_sequence
- kokkosp.parallel_for
- kokkosp.parallel_reduce
- kokkosp.parallel_scan
- kokkosp.parallel_scan_no_fence
- kokkosp.parallel_scan_no_fence_view

- simd.device_math_ops
- simd.device_mask_ops
- simd.device_conversions
- simd.device_shift_ops
- simd.device_condition
- simd.device_gen_ctors
- simd.device_where_expressions
- simd.device_reductions
- simd.device_construction

oneAPI and ArborX

same problem with ArborX:

```
[...]
30: BM_knn_callback_search<ArborX::BVH<Serial>>/50000/20000/10/1/0/0/manual_time
30: <HIP>[ERROR]:
30: UR HIP ERROR:
30:  Value:           209
30:  Name:            hipErrorNoBinaryForGpu
30:  Description:     no kernel image is available for execution on the device
30:  Function:        buildProgram
30:  Source Location: [...]
30:
30: terminate called after throwing an instance of 'sycl::_V1::exception'
30:   what():  The program was built for 1 devices
30: Build program log for 'AMD Instinct MI210':
```

oneAPI and ArborX

Failing in

```
sycl::kernel_id functor_kernel_id
  = sycl::get_kernel_id<decltype(dummy_reduction_lambda)>();
auto kernel_bundle
  = sycl::get_kernel_bundle<sycl::bundle_state::executable>(
                  q.get_context(), std::vector{functor_kernel_id});
sycl::kernel kernel = kernel_bundle.get_kernel(functor_kernel_id);
auto multiple = kernel.get_info<sycl::info::kernel_device_specific::
                                  preferred_work_group_size_multiple>(
            q.get_device());
```

which works in a lot of `Kokkos` unit tests. Replacing with fixed values doesn't help.

AdaptiveCpp on Frontier

Compiling with `AdaptiveCpp` runs into a bunch of problems

- sycl::ctz
- sycl::nan
- sycl::ext::oneapi::experimental::printf
- sycl::kernel_id, sycl::get_kernel_id
- sycl::ext::oneapi::group_local_memory_for_overwrite
- sycl::ext::intel::device_ptr, sycl::ext::intel::host_ptr
- sycl::ext::oneapi::experimental::this_sub_group
- sycl::atomic_fence

AdaptiveCpp on Frontier

Disabling extensions means

- No arbitrary size atomics
- No `TeamPolicy`
- No `Graph` support
- No `Kokkos::Random`

AdaptiveCpp on Frontier

Configuring `AdaptiveCpp` with existing software (+`Boost`)

```
cmake
  -DCMAKE_CXX_COMPILER=hipcc
  -DBOOST_ROOT=~/boost-1-87-install
  -DWITH_SSCP_COMPILER=OFF
  -DWITH_OPENCL_BACKEND=OFF
  -DWITH_LEVEL_ZERO_BACKEND=OFF
  -DDEFAULT_TARGETS=hip:gfx90a
  ..
```

Configuring `Kokkos`

```
cmake
  -DCMAKE_CXX_COMPILER=acpp
  -DKokkos_ENABLE_SYCL=ON
  -DKokkos_ENABLE_ONEDPL=OFF
  -DCMAKE_CXX_FLAGS="--acpp-targets=hip"
  -DCMAKE_BUILD_TYPE=Release
  ..
```

AdaptiveCpp on Frontier

- UnitTest_Serial1 (Failed)
- UnitTest_SYCL1A (Failed)
- UnitTest_SYCL1B (SEGFAULT)
- UnitTest_SYCL2A (Timeout)
- UnitTest_SYCL3 (SEGFAULT)
- UnitTest_SYCLInterOpInit (Failed)
- UnitTest_SYCLInterOpInit_Context (Failed)
- UnitTest_SYCLInterOpStreams (Failed)
- UnitTest_SYCLInterOpStreamsMultiGPU (Timeout)
- UnitTest_Default (SEGFAULT)
- UnitTest_InitializeFinalize (Failed)
- IncrementalTest_SYCL (SEGFAULT)
- ContainersUnitTest_SYCL (Timeout)
- UnitTest_Sort (SEGFAULT)
- UnitTest_Random (Failed)
- UnitTest_StdSet_A (Failed)
- UnitTest_StdSet_B (Failed)
- UnitTest_StdSet_C (Failed)
- UnitTest_StdSet_D (Failed)
- UnitTest_StdSet_E (Failed)
- UnitTest_StdSet_Team_A (SEGFAULT)
- UnitTest_StdSet_Team_B (SEGFAULT)
- UnitTest_StdSet_Team_C (SEGFAULT)
- UnitTest_StdSet_Team_D (SEGFAULT)
- UnitTest_StdSet_Team_E (Timeout)
- UnitTest_StdSet_Team_F (Timeout)
- UnitTest_StdSet_Team_G (Timeout)
- UnitTest_StdSet_Team_H (SEGFAULT)
- UnitTest_StdSet_Team_I (SEGFAULT)
- UnitTest_StdSet_Team_L (SEGFAULT)
- UnitTest_StdSet_Team_M (Timeout)
- UnitTest_StdSet_Team_P (SEGFAULT)
- UnitTest_StdSet_Team_Q (SEGFAULT)

AdaptiveCpp with upstream LLVM

```
make: *** [Makefile:933: Kokkos_IncrementalTest_SYCL] Error 2
[darndt@login05.frontier build_new]$ make Kokkos_IncrementalTest_SYCL
[ 16%] Built target kokkossimd
[ 16%] Building CXX object core/src/CMakeFiles/kokkoscore.dir/impl/Kokkos_Abort.cpp.o
PLEASE submit a bug report to https://github.com/llvm/llvm-project/issues/ and include the crash backtrace, preprocesse
Stack dump:
0. Program arguments: /lustre/orion/ums018/proj-shared/llvm/bin/clang++ -isystem /lustre/orion/ums018/proj-shared/adapt
install/bin/../include/AdaptiveCpp -D__OPENSYCL__ -D__HIPSYCL__ -D__ADAPTIVECPP__ -D__ACPP__ -ffp-contract=fast -
D__ACPP_ENABLE_LLVM_SSCP_TARGET__ -Xclang -disable-O0-optnone -mllvm -acpp-sscp -fplugin=/lustre/orion/ums018/proj-
shared/adaptivecpp-install/bin/../lib/libacpp-clang.so -fpass-plugin=/lustre/orion/ums018/proj-shared/adaptivecpp-
install/bin/../lib/libacpp-clang.so -DKOKKOS_DEPENDENCE -I/ccs/home/darndt/kokkos_adaptivecpp/build_new/core/src -
I/ccs/home/darndt/kokkos_adaptivecpp/core/src -I/ccs/home/darndt/kokkos_adaptivecpp/build_new -I/ccs/home/darndt/kokkos
I/ccs/home/darndt/kokkos_adaptivecpp/tpls/mdspan/include -O3 -DNDEBUG -std=gnu++17 -MD -MT core/src/CMakeFiles/kokkosco
MF CMakeFiles/kokkoscore.dir/impl/Kokkos_Abort.cpp.o.d -o CMakeFiles/kokkoscore.dir/impl/Kokkos_Abort.cpp.o -c /ccs/hom
1. <eof> parser at end of file
2. Optimizer
3. Running pass "hipsycl::compiler::TargetSeparationPass" on module "/ccs/home/darndt/kokkos_adaptivecpp/core/src/impl/
 #0 0x0000000002472848 llvm::sys::PrintStackTrace(llvm::raw_ostream&, int) (/lustre/orion/ums018/proj-shared/llvm/bin/c
 #1 0x00000000024706dc llvm::sys::CleanupOnSignal(unsigned long) (/lustre/orion/ums018/proj-shared/llvm/bin/clang+++0x2
 #2 0x00000000023c4f68 CrashRecoverySignalHandler(int) CrashRecoveryContext.cpp:0:0
[...]
```

Performance comparisons

Performance comparisons

AXPBY - parallel_for

DOT - parallel_reduce

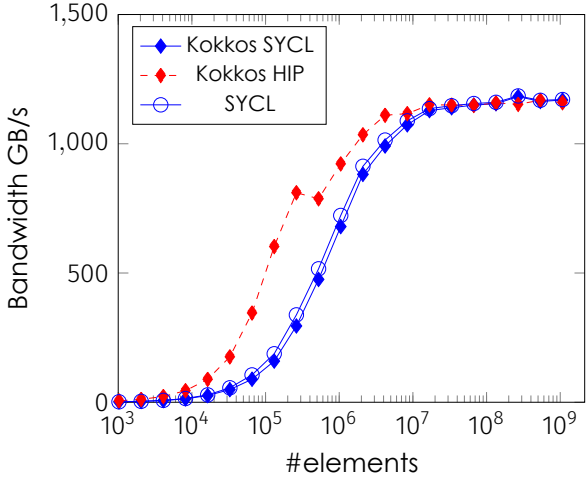
SPMV - TeamPolicy

One node of `Frontier` with AMD Instinct® MI250X GPUs
peak memory bandwidth 3276.8 GB/s/card, 1638.4G GB/s/chiplet

AXPBY benchmark

```
// Kokkos
for (int r = 0; r < R; r++) {
  Kokkos::parallel_for("axpby", N, KOKKOS_LAMBDA(int i) {
   z(i) = alpha*x(i) + beta*y(i);
  });
}

// SYCL
sycl::queue q{sycl::property::queue::in_order()} ;
for (int r = 0; r < R; r++) {
  q.parallel_for(sycl::range<1>(N), [=](sycl::id<1> idx){
    int i = idx;
    z[i] = alpha*x[i] + beta*y[i];
  });
}
```

Achieved effective bandwidth for the AXPBY benchmark

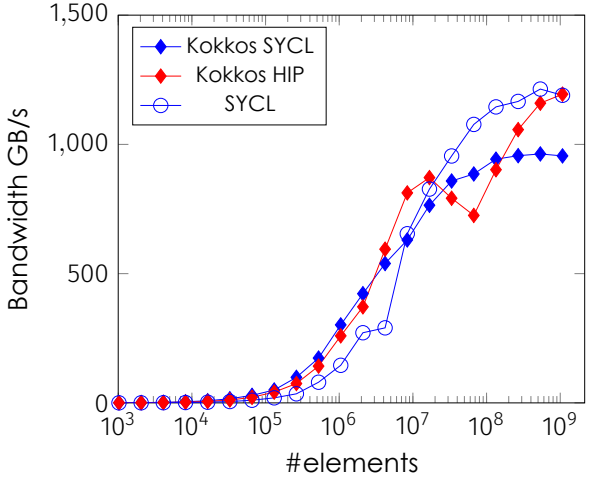


Kokkos::HIP launches from local, global or shared memory based on kernel size.

DOT benchmark

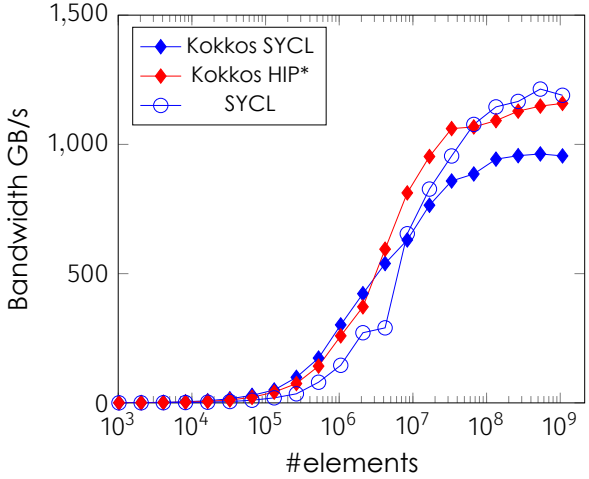
```
// Kokkos
for (int r = 0; r < R; r++) {
  Kokkos::parallel_reduce("dot", N,
    KOKKOS_LAMBDA(int i, double& sum) {sum += x(i) * y(i);},
  result);

// SYCL
sycl::queue q{sycl::property::queue::in_order()} ;
for (int r = 0; r < R; r++) {
  q.parallel_for(sycl::range<1>(N_),
    sycl::reduction(result_ptr, 0., sycl::plus<double>()),
      [=](sycl::id<1> idx, auto&sum) {sum += x[idx] * y[idx];});
  q.memcpy(&result, result_ptr, sizeof(double));
  q.wait();
}
```

Achieved effective bandwidth for the DOT benchmark



Kokkos::SYCL forced to use shuffle reductions instead of local memory reduction.

Achieved effective bandwidth for the DOT benchmark



HIP with 1024 instead of 256 threads per block when #elements $\geq$ 16777216.

SPMV benchmark - Kokkos I

```
int rows_per_team = 32; //optimized for GPU
int team_size = 16;     //optimized for GPU
int vector_size = 4;    //optimized for GPU
int n_teams = (nrows + rows_per_team - 1)/rows_per_team;
using TeamMember = Kokkos::TeamPolicy<>::member_type;
// parallelize over the row blocks
Kokkos::parallel_for("SPMV",
  Kokkos::TeamPolicy<>(n_teams, team_size, vector_size),
    KOKKOS_LAMBDA(const TeamMember &team) {
      int64_t first_row=team.league_rank()*rows_per_team;
      int64_t last_row=first_row + rows_per_team < nrows
        ? first_row + rows_per_team : nrows;
```

SPMV benchmark - Kokkos II

```
// parallelize over rows owned by the team
Kokkos::parallel_for(
  Kokkos::TeamThreadRange(team,first_row,last_row),
    [&](const int64_t row) {
      const int64_t row_start = A.row_ptr(row);
      const int64_t row_length = A.row_ptr(row + 1) - row_start;
      // perform the dot-product of a matrix row with vector
      Kokkos::parallel_reduce(
        Kokkos::ThreadVectorRange(team,row_length),
          [=](const int64_t i, double &sum) {
            sum += A.values(i + row_start) * x(A.col_idx(i + row_start));
          }, y(row));
  });
});
```

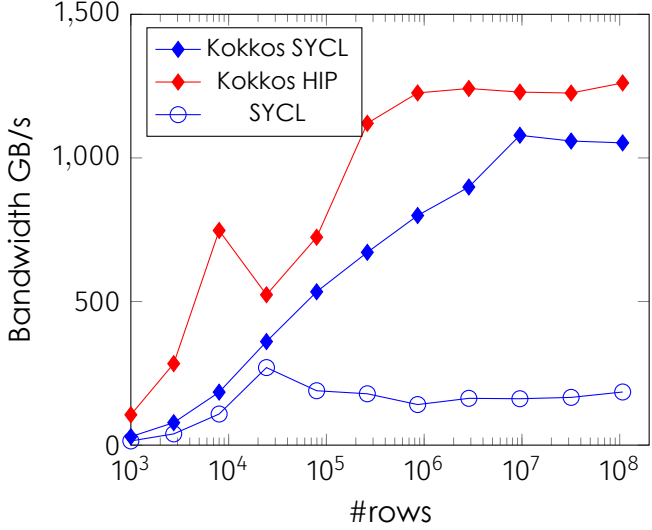
SPMV benchmark - SYCL I

```
int rows_per_team = 32; //optimized for GPU
int team_size = 16;     //optimized for GPU
int n_teams = (nrows + rows_per_team - 1)/rows_per_team;
q.submit([&] (sycl::handler& cgh) {
  // parallelize over the row blocks
  cgh.parallel_for_work_group(sycl::range<1>(n),
    sycl::range<1>(team_size), [=](sycl::group<1> g) {
      int64_t first_row= g.get_group_id(0)*rows_per_team;
      int64_t last_row=first_row + rows_per_team < nrows
        ? first_row + rows_per_team : nrows;
```

SPMV benchmark - SYCL II

```
        // parallelize over rows owned by the team
        g.parallel_for_work_item(
          sycl::range<1>(last_row-first_row),
          [&](sycl::h_item<1> item) {
            int64_t row = item.get_local_id(0)+first_row;
            int64_t row_start = row_ptr[row];
            int64_t row_length = row_ptr[row+1]-row_start;
            double y_row = 0.;
            for (int64_t i = 0; i < row_length; ++i)
              y_row += values[i + row_start] * xp[col_idx[i + row_start]];
            yp[row] = y_row;
      });
    });
  });
```

Achieved effective bandwidth for the SPMV benchmark on the GPU



Summary

Despite some hiccups, `Kokkos::SYCL` on AMD GPUs

- largely works with `oneAPI` for very recent compilers
- still hard to diagnose runtime errors in some cases
- `printf` missing makes debugging very hard
- decent performance, in particular for reductions
- lack of hierarchical parallelism abstraction still problematic
- not practical with `AdaptiveCPP` at this point

Acknowledgments

- This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy.
- This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies program, under contract number DE-AC05-00OR22725.

Questions?

Appendix

parallel_for RangePolicy

```
Kokkos::parallel_for(
  Kokkos::RangePolicy(execution_space, start, end)),
  KOKKOS_LAMBDA(int i) {/*...*/});
```

is mapped to SYCL code as

```
Functor functor;
q.parallel_for(sycl::range<1>(end - begin),
  [=](sycl::id<1> idx) {
    int i = idx + begin;
    functor(i);
});
```

Kokkos iota

```
#include <Kokkos_Core.hpp>
int main() {
  Kokkos::ScopeGuard scope_guard;
  Kokkos::View<int*> view("view", 100);
  Kokkos::parallel_for(100, KOKKOS_LAMBDA(int i){view(i) = i;});
}
```

⇒ Using `sycl::buffer` for `Kokkos::View` not feasible.

Problem:

- `Kokkos::View` is not trivially copyable.
- `sycl::is_device_copyable`?

`sycl::is_device_copyable`

> It is unspecified whether the implementation actually calls the copy constructor, move constructor, copy assignment operator, or move assignment operator of a class declared as *`is_device_copyable_v`* when doing an inter-device copy.
>
> [...]
>
> Likewise, it is unspecified whether the implementation actually calls the destructor for such a class on the device since the destructor must have no effect on the device.

Issue:

- Implementations actually call special member functions[1]
- We need another workaround!

[1] `https://github.com/intel/llvm/issues/5320`

`sycl::is_device_copyable` - Workaround I

```
union TrivialWrapper {
  TrivialWrapper(){};
  TrivialWrapper(const Functor& f) {
    std::memcpy(&m_f, &f, sizeof(m_f)); }
  TrivialWrapper(const TrivialWrapper& other) {
    std::memcpy(&m_f, &other.m_f, sizeof(m_f));
  }
  TrivialWrapper& operator=(const TrivialWrapper& other) {
    std::memcpy(&m_f, &other.m_f, sizeof(m_f));
    return *this;
  }
  ~TrivialWrapper(){};
  Functor m_f;
};
```

`sycl::is_device_copyable` - Workaround II

```
template <typename Functor>
class SYCLFunctionWrapper {
  union TrivialWrapper m_functor;
 public:
  SYCLFunctionWrapper(const Functor& functor, Storage&)
    : m_functor(functor) {}
  const Functor& get_functor() const {
    return m_functor.m_f;
  }
};

template <typename Functor>
struct sycl::is_device_copyable<
    SYCLFunctionWrapper<Functor, Storage, false>>
    : std::true_type {};
```

MDRangePolicy

`MDRangePolicy` maps up to 6 dimensions with tiling to three dimensions in `sycl::nd_range`

```
struct Functor{
  KOKKOS_FUNCTION void operator(
    int i, int j, int k, int l, int m) const {/*...*/}
};
Kokkos::parallel_for(
  Kokkos::MDRangePolicy(execution_space,
    {s0,s1,s2,s3,s4}, {e0,e1,e2,e3,e4}),
  Functor{});
```

TeamPolicy

```
parallel_for("Label", TeamPolicy<>(numberOfTeams, teamSize, vectorLength),
  KOKKOS_LAMBDA (const member_type & teamMember) {
    /* beginning of outer body */
    parallel_for(TeamThreadRange(teamMember, thisTeamsRangeSize),
      [=] (const int indexWithinBatch[, ...]) {
        /* begin middle body */
        parallel_for(ThreadVectorRange(teamMember, thisVectorRangeSize),
            [=] (const int indexVectorRange) {/* inner body */});
        /* end middle body */
      });
    parallel_for(TeamVectorRange(teamMember, someSize),
      [=] (const int indexTeamVector) {/* nested body */});
    /* end of outer body */
  });
```

TeamPolicy

Only policy to allow scratch allocations.

Mappings:

- team → `sycl::group`
- multiple threads → `sycl::subgroup`

Problem:

- thread synchronization requires a barrier on part of a subgroup → `tangle_group` or similar
- communicating address space information for scratch allocations

parallel_reduce

```
double result;
Kokkos::parallel_reduce(
  Kokkos::RangePolicy(execution_space, start, end)),
  KOKKOS_LAMBDA(int i, double& partial_sum) {
    partial_sum += i;
  }, result);
```

doesn't use SYCL's reduction variables. Features:

- simple reductions (sum)
- multiple reductions per parallel construct
- custom reductions with arbitrary value types and reduction operations
- runtime sized array reductions
- pre- and post-callbacks for reductions (`init`, `final`)

parallel_reduce

```
per_thread:
  value& tmp=init(local_tmp);
  for (i in local range)
    functor(i, tmp)
call join for merging values between threads
  in the same workgroup
let one (the last) workgroup merge all results
  from all workgroups
call final(result) on one thread
```

Shuffle-based implementation gives worse results than using local memory on Intel GPUs.

parallel_scan

```
Kokkos::parallel_scan(
  Kokkos::RangePolicy(execution_space, start, end),
  KOKKOS_LAMBDA (const int index, value_type& update,
                 const bool is_final) {
    const value_type local_value = in_data(i);
    // exclusive scan
    if (is_final)
      out_data_exclusive(i) = update;
    update += local_value;
    // inclusive scan
    if (is_final)
      out_data_inclusive(i) = update;
 });
```

parallel_scan

```
first kernel:
  per_thread:
    value& tmp=init(local_tmp);
    for (i in local range)
      functor(i, tmp, /*is_final*/ false)
  call join for implementing a prefix sum
    in the same workgroup
  let the last workgroup compute the prefix sum for the
    totals of all workgroups and store the result
  store intermediate results on each thread
second kernel:
  combine workgroup totals with thread intermediate results
  call the functor again for final result (with final=true)
```

As opposed to `parallel_reduce` a shuffle-based implementation is used.

Appendix

Performance comparisons

AXPBY - parallel_for

DOT - parallel_reduce

SPMV - TeamPolicy

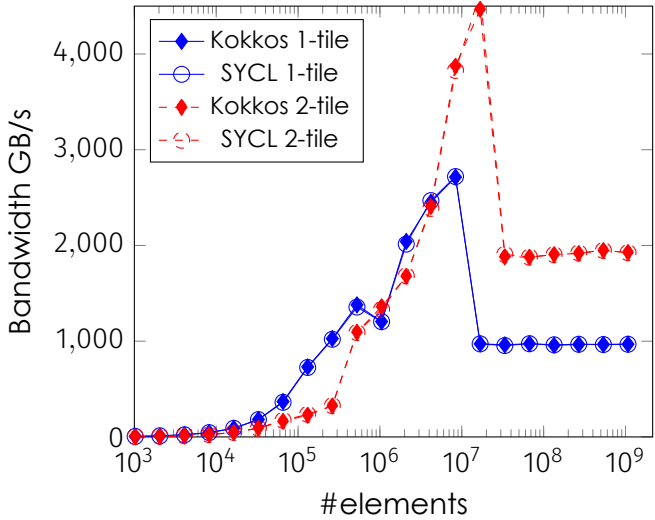
Conjugate Gradient

One node of `Sunspot` with Intel® Data Center GPU Max 1550 GPUs
peak memory bandwidth 3276.8 GB/s (2 tiles)

AXPBY benchmark

```
// Kokkos
for (int r = 0; r < R; r++) {
  Kokkos::parallel_for("axpby", N, KOKKOS_LAMBDA(int i) {
   z(i) = alpha*x(i) + beta*y(i);
  });
}

// SYCL
sycl::queue q{sycl::property::queue::in_order()} ;
for (int r = 0; r < R; r++) {
  q.parallel_for(sycl::range<1>(N), [=](sycl::id<1> idx){
    int i = idx;
    z[i] = alpha*x[i] + beta*y[i];
  });
}
```

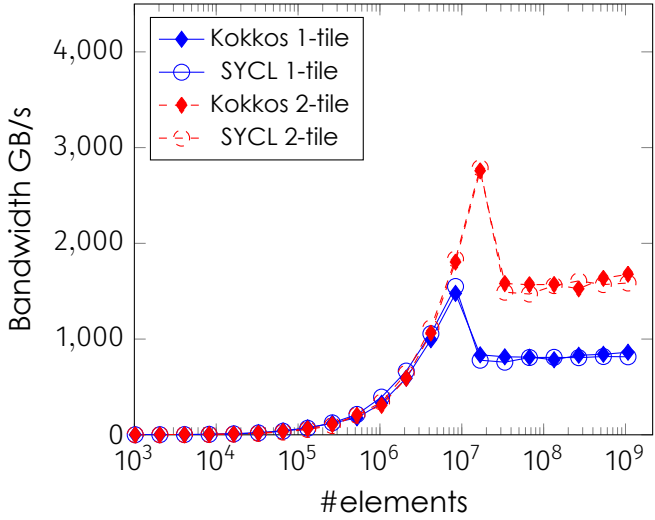
Achieved effective bandwidth for the AXPBY benchmark



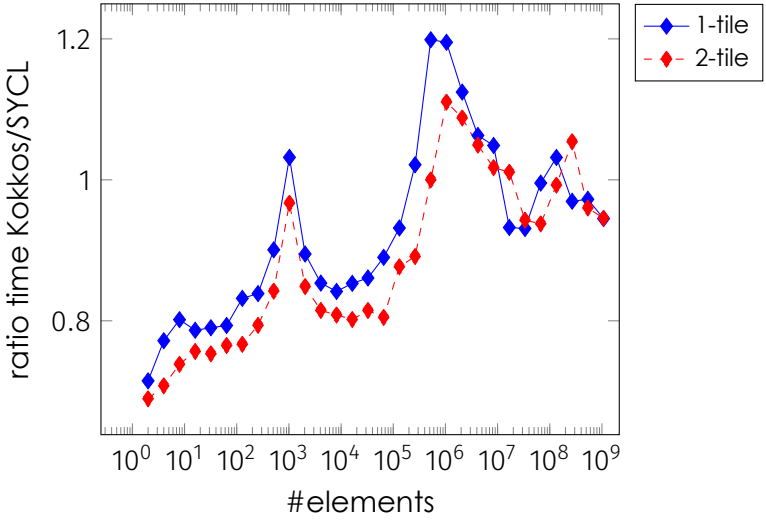
DOT benchmark

```
// Kokkos
for (int r = 0; r < R; r++) {
  Kokkos::parallel_reduce("dot", N,
    KOKKOS_LAMBDA(int i, double& sum) {sum += x(i) * y(i);},
  result);

// SYCL
sycl::queue q{sycl::property::queue::in_order()} ;
for (int r = 0; r < R; r++) {
  q.parallel_for(sycl::range<1>(N_),
    sycl::reduction(result_ptr, 0., sycl::plus<double>()),
      [=](sycl::id<1> idx, auto&sum) {sum += x[idx] * y[idx];});
  q.memcpy(&result, result_ptr, sizeof(double));
  q.wait();
}
```

Achieved effective bandwidth for the DOT benchmark



Run time ratio for DOT between Kokkos and **SYCL**



SPMV benchmark - Kokkos I

```
int rows_per_team = 128; //optimized for GPU
int team_size = 128;     //optimized for GPU
int vector_size = 8;    //optimized for GPU
int n_teams = (nrows + rows_per_team - 1)/rows_per_team;
using TeamMember = Kokkos::TeamPolicy<>::member_type;
// parallelize over the row blocks
Kokkos::parallel_for("SPMV",
  Kokkos::TeamPolicy<>(n_teams, team_size, vector_size),
    KOKKOS_LAMBDA(const TeamMember &team) {
      int64_t first_row=team.league_rank()*rows_per_team;
      int64_t last_row=first_row + rows_per_team < nrows
        ? first_row + rows_per_team : nrows;
```

SPMV benchmark - Kokkos II

```
// parallelize over rows owned by the team
Kokkos::parallel_for(
  Kokkos::TeamThreadRange(team,first_row,last_row),
    [&](const int64_t row) {
      const int64_t row_start = A.row_ptr(row);
      const int64_t row_length = A.row_ptr(row + 1) - row_start;
      // perform the dot-product of a matrix row with vector
      Kokkos::parallel_reduce(
        Kokkos::ThreadVectorRange(team,row_length),
          [=](const int64_t i, double &sum) {
            sum += A.values(i + row_start) * x(A.col_idx(i + row_start));
          }, y(row));
  });
});
```

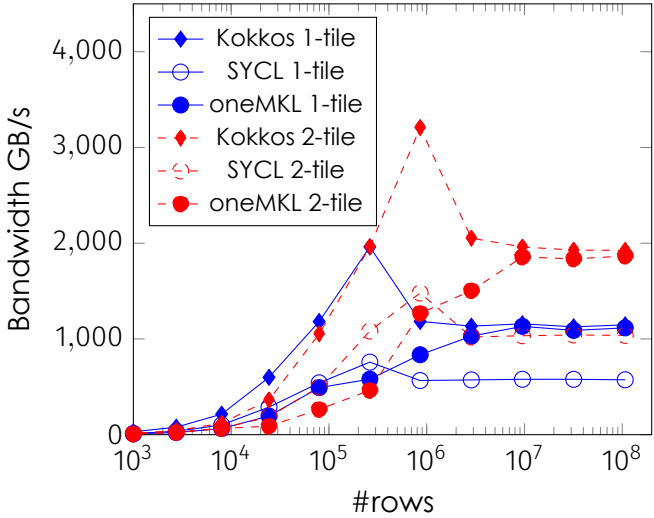
SPMV benchmark - SYCL I

```
int rows_per_team = 1024; //optimized for GPU
int team_size = 1024;     //optimized for GPU
int n_teams = (nrows + rows_per_team - 1)/rows_per_team;
q.submit([&] (sycl::handler& cgh) {
  // parallelize over the row blocks
  cgh.parallel_for_work_group(sycl::range<1>(n),
    sycl::range<1>(team_size), [=](sycl::group<1> g) {
      int64_t first_row= g.get_group_id(0)*rows_per_team;
      int64_t last_row=first_row + rows_per_team < nrows
        ? first_row + rows_per_team : nrows;
```

SPMV benchmark - SYCL II

```
        // parallelize over rows owned by the team
        g.parallel_for_work_item(
          sycl::range<1>(last_row-first_row),
          [&](sycl::h_item<1> item) {
            int64_t row = item.get_local_id(0)+first_row;
            int64_t row_start = row_ptr[row];
            int64_t row_length = row_ptr[row+1]-row_start;
            double y_row = 0.;
            for (int64_t i = 0; i < row_length; ++i)
              y_row += values[i + row_start] * xp[col_idx[i + row_start]];
            yp[row] = y_row;
      });
    });
  });
```

Achieved effective bandwidth for the SPMV benchmark on the GPU



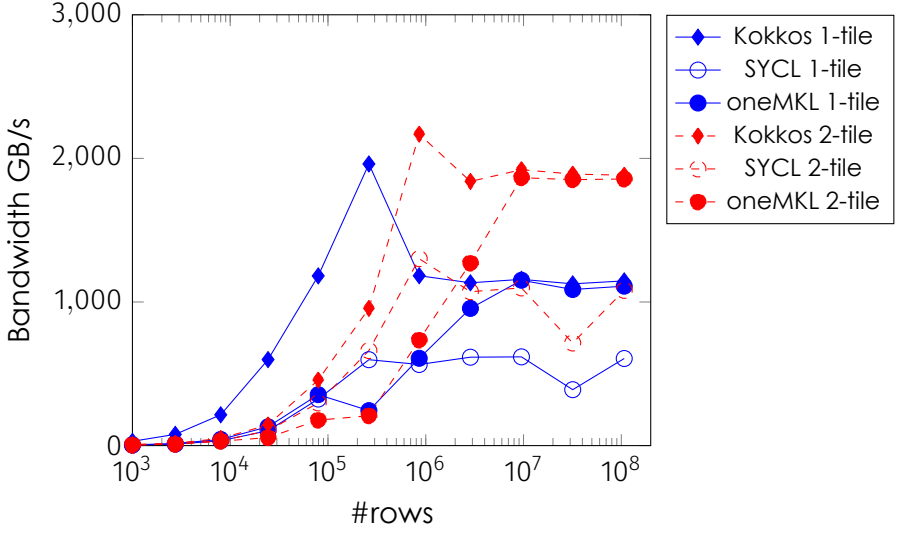
CG benchmark I

```
for (int64_t k = 1; k <= max_iter && normr > tolerance; ++k) {
  if (k == 1) {
    axpby(p, one, r, zero, r);
  } else {
    oldrtrans = rtrans;
    rtrans = dot(r, r);
    double beta = rtrans / oldrtrans;
    axpby(p, one, r, beta, p);
  }
  normr = std::sqrt(rtrans);
  double alpha = 0;
  double p_ap_dot = 0;
  spmv(Ap, A, p);
```

CG benchmark II

```
    p_ap_dot = dot(Ap, p);
    if (p_ap_dot < brkdown_tol) {
      if (p_ap_dot < 0) {
        std::cerr << "numerical breakdown!\n";
        return num_iters;
      } else
        brkdown_tol = 0.1 * p_ap_dot;
    }
    alpha = rtrans / p_ap_dot;
    axpby(x, one, x, alpha, p);
    axpby(r, one, r, -alpha, Ap);
    num_iters = k;
  }
```

Achieved effective bandwidth for the CG benchmark on the GPU



Summary

Despite some hiccups, SYCL/DPC++

- integration was pretty smooth
- works well on Intel GPUs (Aurora)
- works much better than `OpenMPTarget`
- has better support for newer C++ features than `nvcc`

We still rely on many extensions, though.

Questions?

Acknowledgments

- This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy.

- This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.

- This work was done on a pre-production supercomputer with early versions of the Aurora software development kit.