IWOCL 2024

The 12th International Workshop on OpenCL and SYCL

Experiences with Implementing Kokkos' SYCL Backend

Daniel Arndt, Oak Ridge National Laboratory

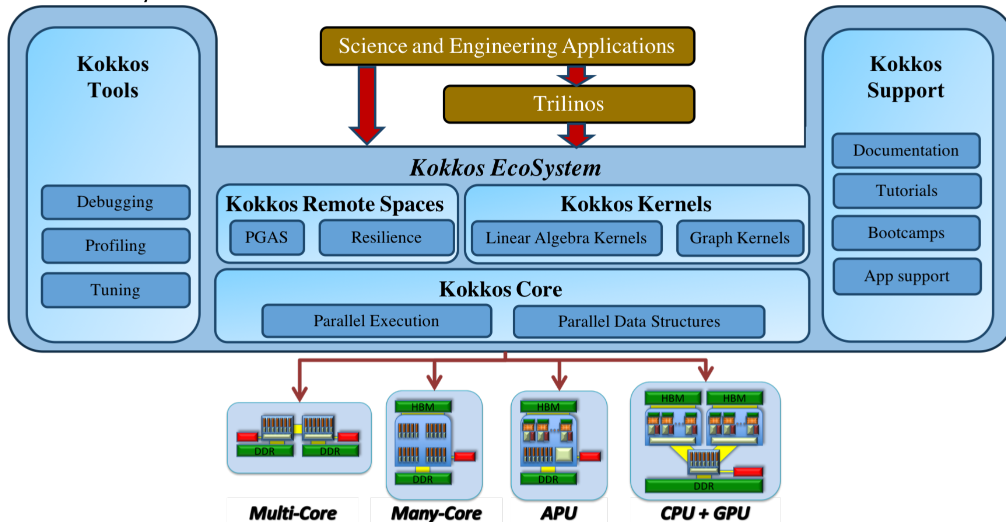
Daniel Arndt, Damien Lebrun-Grandie, Oak Ridge National Laboratory and Christian Trott, Sandia National Laboratories.

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

Oak Ridge National Laboratory

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)
- ...

Feature Status Kokkos+SYCL targeting Aurora/Intel GPUs

Feature complete apart from

- `WorkGraphPolicy`
- `Tasks`
- `Graphs`
- Virtual functions/function pointer

OAK RIDGE National Laboratory

Kokkos Core Functionalities, Mapping to SYCL

Constructs

- `parallel_for` $\longrightarrow$ `sycl::parallel_for`
- `parallel_reduce` $\longrightarrow$ `sycl::parallel_for`
- `parallel_scan` $\longrightarrow$ `sycl::parallel_for`

Policies

- `RangePolicy` $\longrightarrow$ `sycl::range`
- `MDRangePolicy` $\longrightarrow$ `sycl::nd_range`
- `TeamPolicy` $\longrightarrow$ `sycl::nd_range`

Memory

- `View` $\longrightarrow$ `sycl::malloc/sycl::free`

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);
});
```


Oak Ridge National Laboratory

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`?

OAK RIDGE National Laboratory

`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`

Oak Ridge National Laboratory

`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;
};
```


Oak Ridge National Laboratory

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 */
  });
```


OAK RIDGE National Laboratory

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`)

Oak Ridge National Laboratory

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;
 });
```


Oak Ridge National Laboratory

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.

Oak Ridge National Laboratory

Performance comparisons

AXPBY - parallel_for

DOT - parallel_reduce

SPMV - TeamPolicy

Conjugate Gradient

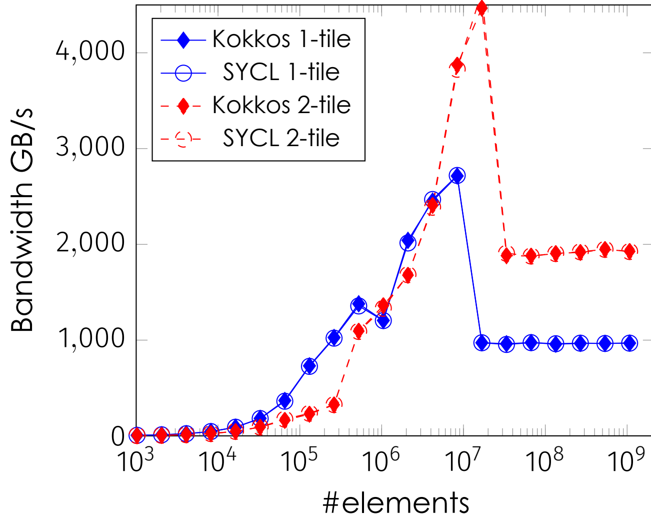
One node of `Sunspot` with Intel® Data Center GPU Max 1550 GPUs
peak memory bandwidth 3276.8 GB/s (2 tiles)

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];
  });
}
```


Oak Ridge National Laboratory

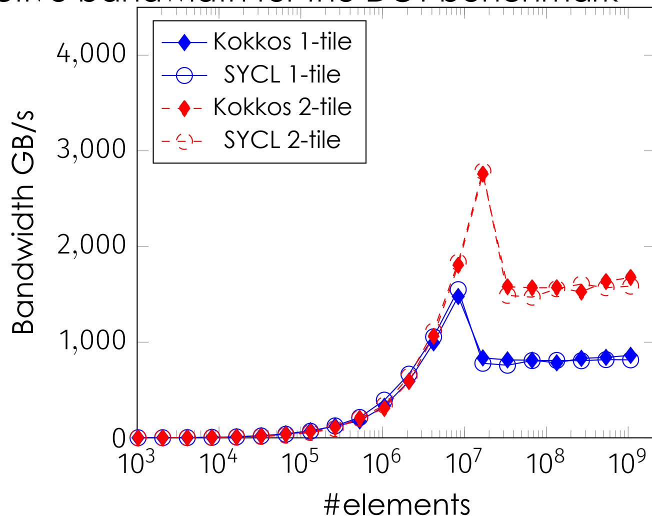
Achieved effective bandwidth for the AXPBY benchmark



DOT benchmark

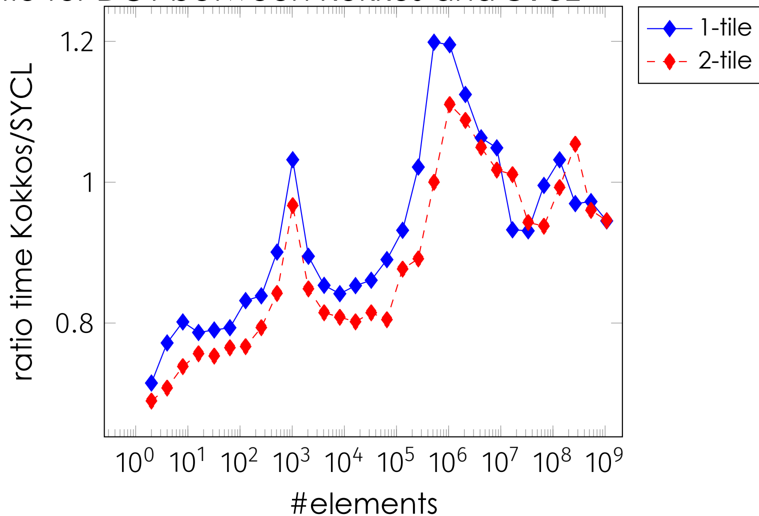
```
// Kokkos
for (int r = 0; r < R; r++) {
  Kokkos::parallel_reduce("dot", N,
    KOKKOS_LAMBDA(int i, double& sum) {sum += x(i) * y(i);},
  result);

// 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



OAK RIDGE National Laboratory

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



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));
  });
});
```


OAK RIDGE National Laboratory

SPMV benchmark - SYCL I

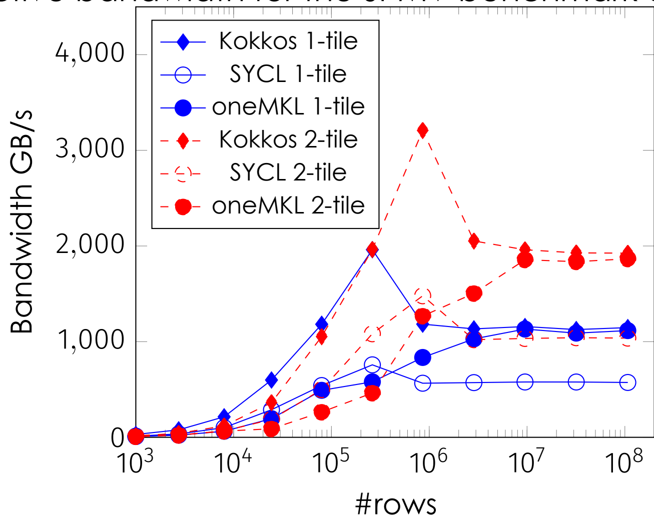
```
int rows_per_team = 32; //optimized for GPU
int team_size = 16;     //optimized for GPU
int n_teams = (nrows + rows_per_team - 1)/rows_per_team;
q.submit([&] (sycl::handler& cgh) {
  // parallelize over the row blocks
  cgh.parallel_for_work_group(sycl::range<1>(n),
    sycl::range<1>(team_size),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


OAK RIDGE National Laboratory

Achieved effective bandwidth for the SPMV benchmark on the GPU



CG benchmark I

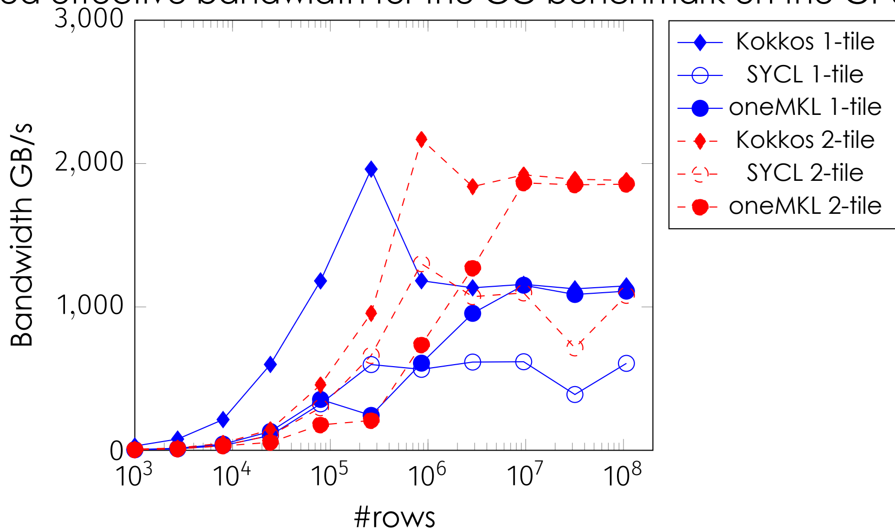
```
for (int64_t k = 1; k <= max_iter && normr > tolerance; ++k) {
  if (k == 1) {
    axpby(p, one, r, zero, r);
  } else {
    oldrtrans = rtrans;
    rtrans = dot(r, r);
    double beta = rtrans / oldrtrans;
    axpby(p, one, r, beta, p);
  }
  normr = std::sqrt(rtrans);
  double alpha = 0;
  double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


OAK RIDGE National Laboratory

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.

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::device_image_scope`
- `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`