Oak Ridge National Laboratory

deal.II and GPUs

Daniel Arndt
Bruno Turcksin

August 14, 2024

11th deal.II Users and Developers Workshop

U.S. DEPARTMENT OF ENERGY

deal.II and GPUs

Places in `deal.II` that (can) use GPUs

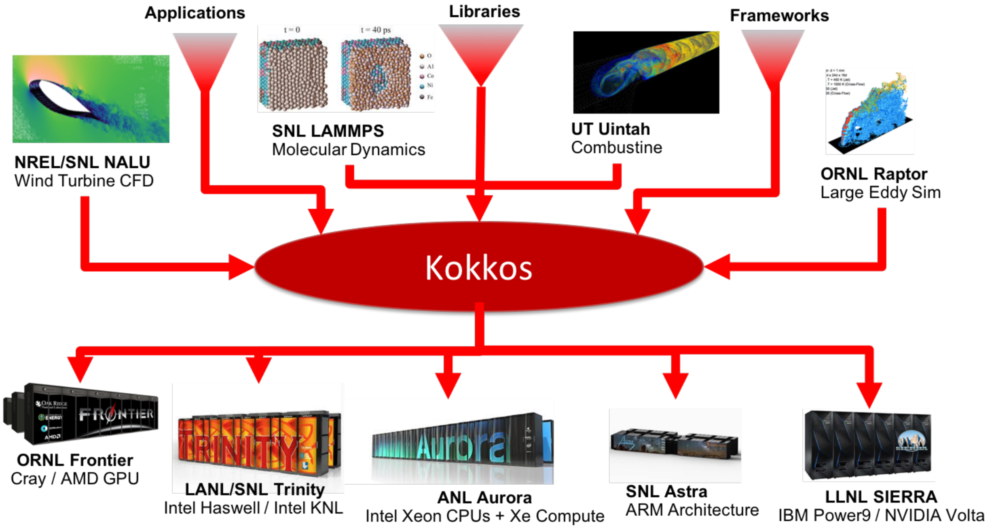
- `PETSc` (TBD)
- `Trilinos`
- `Portable::MatrixFree`
- `LinearAlgebra::distributed::Vector`
- `CUDAWrappers` (to be removed in the next release)

Most of these (except the latter two) use `Kokkos` under the hood!

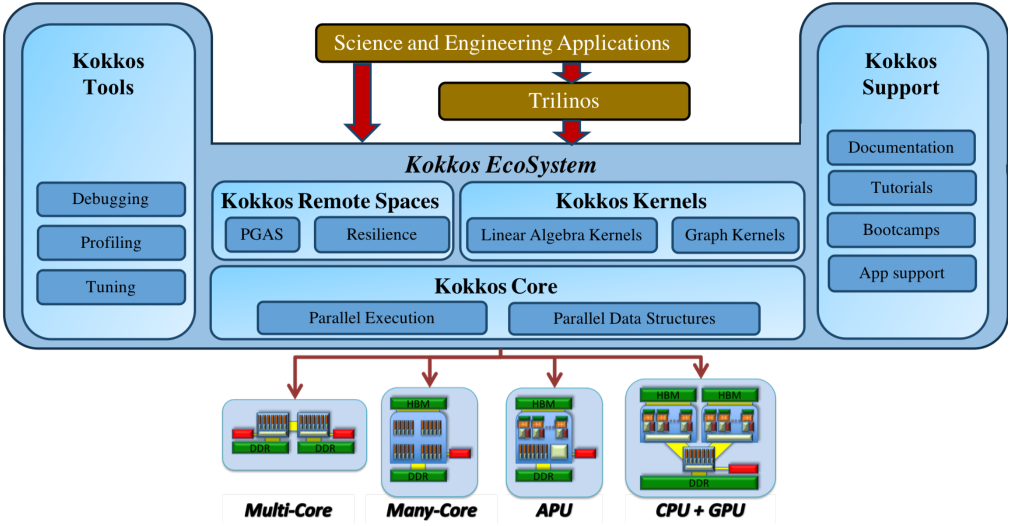
What is Kokkos?

- 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 communixty
  - Maintained and developed at `https://github.com/kokkos`
  - Hundreds of users at many large institutions

Kokkos as Portability Layer



Kokkos EcoSystem



kokkos

Los Alamos
NATIONAL LABORATORY
EST. 1943

Sandia
National
Laboratories

Argonne
NATIONAL LABORATORY

OAK RIDGE
National Laboratory

BERKELEY LAB

CSCS

Kokkos' basic capabilities:

- Simple 1D data parallel computational patterns
- Deciding where code is run and where data is placed
- Managing data access patterns for performance portability
- Multidimensional data parallelism

Kokkos' advanced capabilities:

- Thread safety, thread scalability, and atomic operations
- Hierarchical patterns for maximizing parallelism
- Task-based programming with Kokkos

Kokkos' tools and Kernels:

- How to profile, tune and debug Kokkos code
- Interacting with Python and Fortran
- Using KokkosKernels math library

Kokkos Resources

Online Resources:

- `https://github.com/kokkos`:
  - Primary Kokkos GitHub Organization
- `https://github.com/kokkos/kokkos-tutorials/wiki/Kokkos-Lecture-Series`:
  - Slides, recording and Q&A for the Lectures
- `https://kokkos.github.io/kokkos-core-wiki`:
  - Kokkos Core Wiki with API documentation
- `https://kokkosteam.slack.com`:
  - Slack channel for Kokkos.
  - Please join: fastest way to get your questions answered.
  - Can whitelist domains, or invite individual people.

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;});
}
```

`Kokkos::View`

- managed, multi-dimensional, reference-counted data container, or
- unmanaged, multi-dimensional accessor for external data
- motivated C++23 `std::mdspan`

parallel_reduce

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

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_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;
 });
```

MDRangePolicy

`MDRangePolicy` allows using up to a 6-dimensional index space with tiling

```
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

deal.II and GPUs

Places in `deal.II` that (can) use GPUs

- `PETSc` (TBD)
- `Trilinos`
- `Portable::MatrixFree`
- `LinearAlgebra::distributed::Vector`
- `CUDAWrappers` (to be removed in the next release)

All of these (except the latter) use `Kokkos` under the hood!

Oak Ridge National Laboratory

Trilinos - Tpetra

Added support for Tpetra in this release (Jan-Philip Thiele and Sebastian Kinnewig)

- `Trilinos::TpetraWrappers::Vector<Number, MemorySpace>`
- `Trilinos::TpetraWrappers::SparseMatrix<Number, MemorySpace>`
- `Trilinos::TpetraWrappers::Solver*<Number, MemorySpace>`

Uses `DualView`: Memory is duplicated for `MemorySpace::Default`. All operations happen under the hood.

Standalone linear algebra

`LinearAlgebra::distributed::Vector`

- `LinearAlgebra::distributed::Vector<Number, MemorySpace>`

Memory allocation only for the requested memory space, `MemorySpace::Host` (default) and `MemorySpace::Default`. Requires moving memory to and from the device explicitly.

Assembly in deal.II

Assembly approaches in `deal.II`

- Manually write the assembly of the matrix
- `MatrixCreator`
- `MeshWorker/MeshLoop`
- `MatrixFree`
- `MatrixFreeOperators` Idea:
  - Only overload write (cell) operator (`apply_add`),
  - `MatrixFreeOperators` takes care of the rest

  Similar to `MeshWorker`.

All of them only use CPUs

SPMV matrix-based vs. matrix-free

matrix-based

$$A = \sum_k P_k^T A_k P_k$$

$$v = Au$$

Separate steps for

- Assembly
- matrix-vector product

bandwidth limited: <1 Flop/Byte

matrix-free

$$v = A(u) = \sum_k^{ncells} P_k^T A_k P_k u$$

- Extract local elements: $u_k = P_k u$
- Compute the matrix-vector product on the fly: $v_k = A_k u_k$
- Add to global vector: $v += P_k^T v_k$

OAK RIDGE National Laboratory

Matrix-vector product on cell, Laplacian

$$(A_K u_K)_j = \int_K \nabla_x \phi \cdot \nabla_x u^h \, dx \approx \sum_q w_q \det J_q \nabla \phi_j \cdot \nabla u^h \Big|_{x=x_q}$$

$$= \sum_q \nabla_\eta \phi_j J_q^{-1} (w_q \det J_q) J_q^{-T} \sum_i \nabla_\eta \phi_i u_{K,i} \Big|_{x=x_q}$$

- Compute unit cell gradients $\sum_i \nabla_\eta \phi_i u_{K,i}$ at all quadrature points
- At each quadrature point, apply geometry J_q^{-T}, multiply by quadrature weight and Jacobian determinant $w_q \det J_q$, and apply geometry for test function J_q^{-1}
- Test by unit cell gradients of all basis functions and sum over quadrature points

Sum factorization

$$\sum_i \phi_i(x_q) u_{K,i} = \sum_{i_z} \phi_{i_z}(x_{q_z}) \sum_{i_y} \phi_{i_y}(x_{q_y}) \sum_{i_x} \phi_{i_x}(x_{q_x}) u_{K,i}$$

Take advantage of the tensor product structure of shape functions and quadrature points since $\xi_q = \xi_{q_x}, \xi_{q_y}, \xi_{q_z}$ and $\phi_{x,y,z} = \phi_x \phi_y \phi_z$

$$S = \nabla_\eta \phi_i \big|_{\xi_q} = \begin{bmatrix} D_z \otimes D_y \otimes S_x \\ D_z \otimes S_y \otimes D_x \\ S_z \otimes D_y \otimes D_x \end{bmatrix}$$

with

$$D_x = \phi_x(\xi_{q_x})$$
$$S_x = \nabla_\eta \phi_x(\xi_{q_x})$$

Evaluation costs reduce from $\mathcal{O}((p+1)^{2d})$ to $\mathcal{O}(d(p+1)^{d+1})$.

History of Portable MatrixFree in deal.II

- introduced to deal.II with `https://github.com/dealii/dealii/pull/4293` (deal.II 9.0.0)
- decided to switch to Kokkos in `https://github.com/dealii/dealii/pull/15200` last year
- renamed in `https://github.com/dealii/dealii/pull/16497` this year

Portable MatrixFree limitations

- Only `FE_Q` elements, continuous Galerkin
- no face terms
- only one component
- limited optimizations, even-odd, cartesian meshes
- degree coupled with # quadrature points

Portable MatrixFree implementation

Kokkos implementation

- Uses TeamPolicy for looping over the cells (coloring possible), every team gets a cell batch
- scratch memory for working values and gradients buffers
- TeamThreadRange for tensor contractions on every cell, parallelize over quadrature points
- TeamThreadMDRange for newer Kokkos

deal.II tutorial: step-64

In this example, we consider the Helmholtz problem

$$-\nabla \cdot \nabla u + a(\mathbf{x})u = 1,$$
$$u = 0 \quad \text{on } \partial\Omega$$

where $a(\mathbf{x})$ is a variable coefficient.

We choose as domain $\Omega = [0,1]^3$ and $a(\mathbf{x}) = \frac{10}{0.05+2\|\mathbf{x}\|^2}$. Since the coefficient is symmetric around the origin but the domain is not, we will end up with a non-symmetric solution.

step-64

```
template <int dim, int fe_degree>
  void HelmholtzOperator<dim, fe_degree>::vmult(
    LinearAlgebra::distributed::Vector<double, MemorySpace::Default>       &dst,
    const LinearAlgebra::distributed::Vector<double, MemorySpace::Default> &src)
    const
  {
    dst = 0.;
    LocalHelmholtzOperator<dim, fe_degree> helmholtz_operator(
      coef.get_values());
    mf_data.cell_loop(helmholtz_operator, src, dst);
    mf_data.copy_constrained_values(src, dst);
  }
```

step-64

```
template <int dim, int fe_degree>
DEAL_II_HOST_DEVICE void LocalHelmholtzOperator<dim, fe_degree>::operator()(
  const unsigned int                                         cell,
  const typename Portable::MatrixFree<dim, double>::Data   *gpu_data,
  Portable::SharedData<dim, double>                        *shared_data,
  const double                                             *src,
  double                                                   *dst) const
{
  Portable::FEEvaluation<dim, fe_degree, fe_degree + 1, 1, double> fe_eval(
    gpu_data, shared_data);
  fe_eval.read_dof_values(src);
  fe_eval.evaluate(EvaluationFlags::values | EvaluationFlags::gradients);
  fe_eval.apply_for_each_quad_point(
    HelmholtzOperatorQuad<dim, fe_degree>(gpu_data, coef, cell));
  fe_eval.integrate(EvaluationFlags::values | EvaluationFlags::gradients);
  fe_eval.distribute_local_to_global(dst);
}
```

step-64

```
template <int dim, int fe_degree, int n_q_points_1d, int n_components_, typename Number
DEAL_II_HOST_DEVICE void
FEEvaluation<dim, fe_degree, n_q_points_1d, n_components_, Number>::
  read_dof_values(const Number *src)
{
  Kokkos::parallel_for(
    Kokkos::TeamThreadRange(shared_data->team_member, n_q_points),
      [&](const int &i) {
        shared_data->values(i) = src[data->local_to_global(cell_id, i)];
      });
  shared_data->team_member.team_barrier();

  internal::resolve_hanging_nodes<dim, fe_degree, false>(
    shared_data->team_member,
    data->constraint_weights,
    data->constraint_mask(cell_id),
    shared_data->values);
}
```

Generic - evaluate values and gradients in 3D

```
template <int direction, bool dof_to_quad, bool add, bool in_place,
          typename ViewTypeIn, typename ViewTypeOut>
DEAL_II_HOST_DEVICE void values(const ViewTypeIn in, ViewTypeOut out) const; //apply

values<0, true, false, true>(u, u);
team_member.team_barrier();
values<1, true, false, true>(u, u);
team_member.team_barrier();
values<2, true, false, true>(u, u);
team_member.team_barrier();

gradients<0, true, false, false>(u, Kokkos::subview(grad_u, Kokkos::ALL, 0));
gradients<1, true, false, false>(u, Kokkos::subview(grad_u, Kokkos::ALL, 1));
gradients<2, true, false, false>(u, Kokkos::subview(grad_u, Kokkos::ALL, 2));
```

Generic

```
typename ViewTypeOut>
DEAL_II_HOST_DEVICE void apply_3d(
  const Kokkos::TeamPolicy<...>::member_type &team_member,
  const Kokkos::View<Number *, ...> shape_data, const ViewTypeIn in, ViewTypeOut out)
    { [...]
      Number t[n_q_points] = {};
      auto thread_policy = Kokkos::TeamThreadMDRange<Kokkos::Rank<3>, TeamType>(
        team_member, n_q_points_1d, n_q_points_1d, n_q_points_1d);
      Kokkos::parallel_for(
        thread_policy, [&](const int i, const int j, const int q) {
          const int q_point = ...;
          for (int k = 0; k < n_q_points_1d; ++k) {
              const unsigned int shape_idx  = ...;
              const unsigned int source_idx = ...;
              t[q_point] += shape_data[shape_idx] * in(source_idx);
            }
        }); [...]
    }
```

step-64

```
template <int dim, int fe_degree>
DEAL_II_HOST_DEVICE void HelmholtzOperatorQuad<dim, fe_degree>::operator()(
  Portable::FEEvaluation<dim, fe_degree, fe_degree + 1, 1, double> *fe_eval,
  const int q_point) const
{
  const unsigned int pos =
    gpu_data->local_q_point_id(cell, n_q_points, q_point);

  fe_eval->submit_value(coef[pos] * fe_eval->get_value(q_point), q_point);
  fe_eval->submit_gradient(fe_eval->get_gradient(q_point), q_point);
}
```

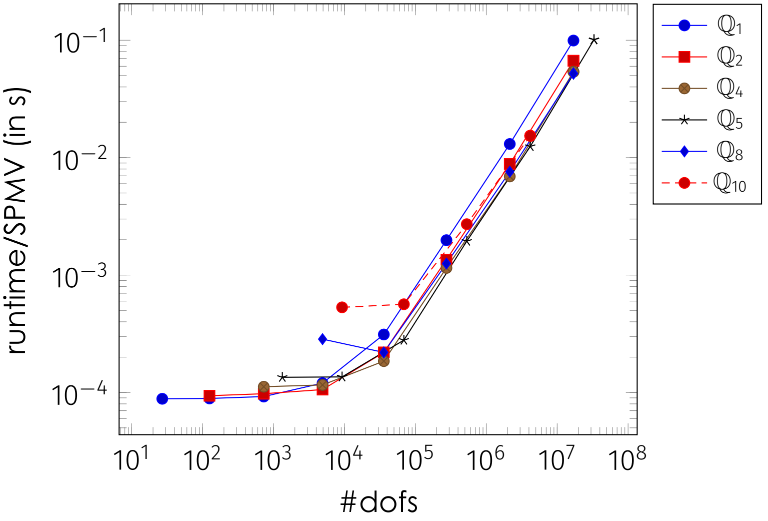
Generic integrate values and gradients in 3D

```
template <int direction, bool dof_to_quad, bool add, bool in_place,
          typename ViewTypeIn, typename ViewTypeOut>
DEAL_II_HOST_DEVICE void values(const ViewTypeIn in, ViewTypeOut out) const; // apply

gradients<2, false, true, false>(Kokkos::subview(grad_u, Kokkos::ALL, 2), u);
team_member.team_barrier();
gradients<1, false, true, false>(Kokkos::subview(grad_u, Kokkos::ALL, 1), u);
team_member.team_barrier();
gradients<0, false, true, false>(Kokkos::subview(grad_u, Kokkos::ALL, 0), u);
team_member.team_barrier();

values<2, false, false, true>(u, u);
team_member.team_barrier();
values<1, false, false, true>(u, u);
team_member.team_barrier();
values<0, false, false, true>(u, u);
team_member.team_barrier();
```

step-64

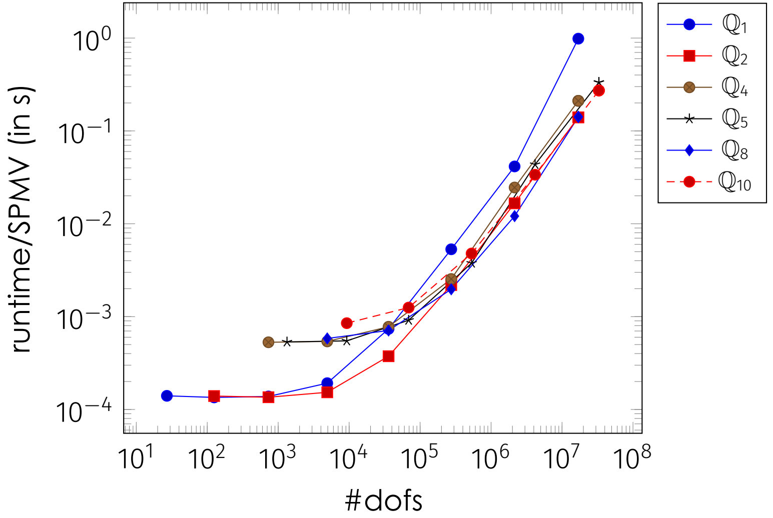
```
template <int dim, int fe_degree, int n_q_points_1d, int n_components_, typename Number>
 DEAL_II_HOST_DEVICE void
 FEEvaluation<dim, fe_degree, n_q_points_1d, n_components_, Number>::
   distribute_local_to_global(Number *dst) const
 {
   internal::resolve_hanging_nodes<dim, fe_degree, true>(
     shared_data->team_member, data->constraint_weights,
     data->constraint_mask(cell_id), shared_data->values);

   if (data->use_coloring) {
       Kokkos::parallel_for(
         Kokkos::TeamThreadRange(shared_data->team_member, n_q_points),
           [&](const int &i) {
             dst[data->local_to_global(cell_id, i)] += shared_data->values(i);
           });
     }
   else { [...] }
 }
```

Performance results on Polaris - NVIDIA A100

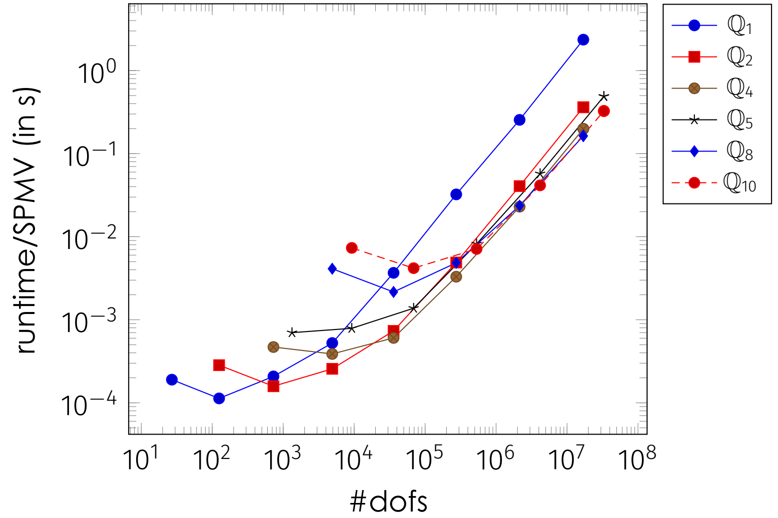


OAK RIDGE National Laboratory

Performance results on Frontier - AMD MI 250X, 1 GCD



Performance results on Sunspot - Intel GPU Max 1550, 1 tile



Summary

- GPUs can be used without changing code using `Tpetra`.
- Portable matrix-free implementation: we can run on CPUs, NVIDIA, AMD, and Intel GPUs using the same code
- Performance tuning needed, in particular for Intel GPUs
- Need to add more capabilities that the CPU version provides

Questions?

Acknowledgments

- This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy.
- This work was funded by S4PST and OLCF through Kokkos Support funding.

OAK RIDGE National Laboratory

Cuda: Performance, A100

Results from `bytes_and_flops(TeamPolicy)`

scalar	Bandwidth	Compute	Cache
`float`	1251 GiB/s	14280 GFlop/s	3762 GiB/s
`double`	1267 GiB/s	7592 GFlop/s	6938 GiB/s
`int32_t`	1222 GiB/s	18457 GFlop/s	4684 GiB/s
`int64_t`	1267 GiB/s	3778 GFlop/s	6895 GiB/s

- Peak FP64 Vector: 19.5 TFLOPS
- Memory Bandwidth: 1.6 TB/sec
- Cache Size: 192KB (per SM)/40 MB

HIP: Performance MI250, one GCD

Results from `bytes_and_flops` (`TeamPolicy`)

scalar	Bandwidth	Compute	Cache
`float`	1160 GiB/s	20544 GFlop/s	2756 GiB/s
`double`	1140 GiB/s	19320 GFlop/s	2883 GiB/s
`int32_t`	1150 GiB/s	20194 GFlop/s	2757 GiB/s
`int64_t`	1140 GiB/s	4979 GFlop/s	2865 GiB/s

- Peak FP64 Vector: 23.95 TFLOPS
- Memory Bandwidth: 1.6 TB/sec
- Cache Size: 16KB (per CU)/16 MB

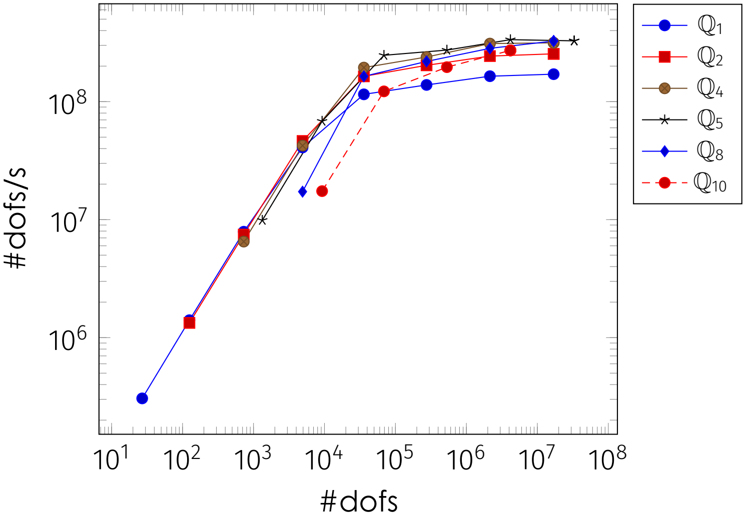
SYCL: Performance Intel Data Center GPU Max 1550, one tile

Results from `bytes_and_flops(TeamPolicy)`

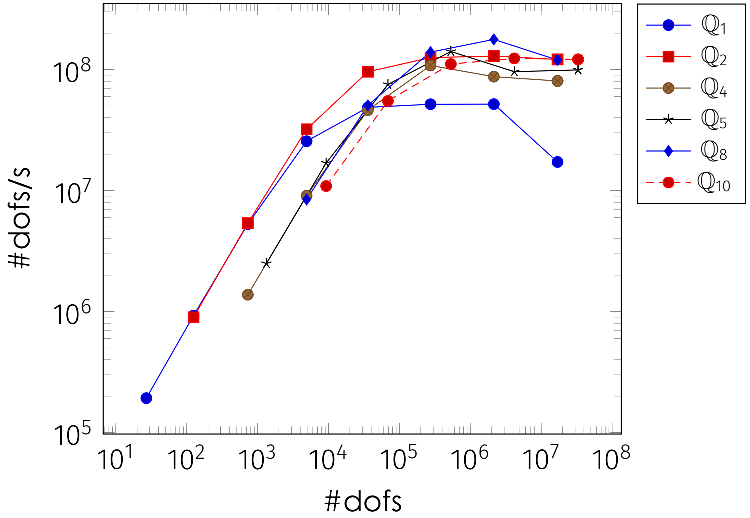
scalar	Bandwidth	Compute	Cache
`float`	1002 GiB/s	17484 GFlop/s	4973 GiB/s
`double`	960 GiB/s	8746 GFlop/s	6928 GiB/s
`int32_t`	1007 GiB/s	6108 GFlop/s	4714 GiB/s
`int64_t`	958 GiB/s	982 GFlop/s	4715 GiB/s

- Peak FP64 Vector: 22.9 TFLOPS/tile
- Memory Bandwidth: 1.6 TB/sec/tile
- Cache Size: 128KB (per work group)/408 MB

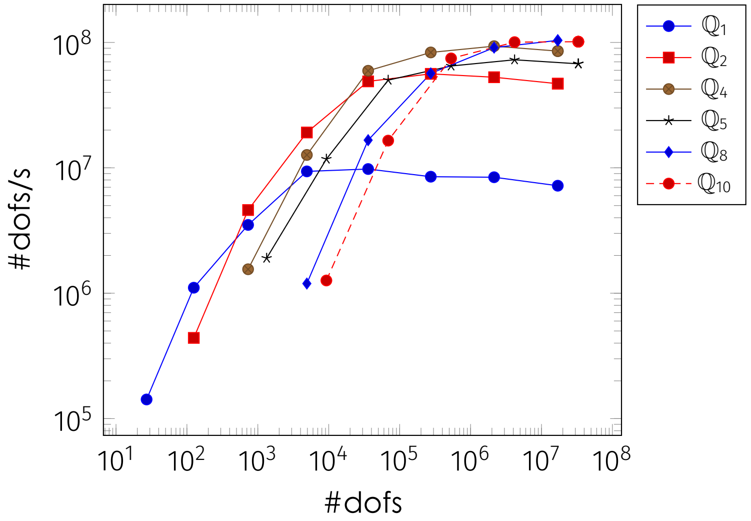
Performance results on Polaris - NVIDIA A100



Performance results on Frontier - AMD MI 250X, 1 GCD



Performance results on Sunspot - Intel GPU Max 1550, 1 tile


10^5
10^6
10^7
10^8
#dofs/s
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
#dofs
$\mathbb{Q}_1$
$\mathbb{Q}_2$
$\mathbb{Q}_4$
$\mathbb{Q}_5$
$\mathbb{Q}_8$
$\mathbb{Q}_{10}$