THURSDAY DECEMBER 14TH, 2023

TEAM-LEVEL MDRANGE POLICIES

NEVIN LIBER
Argonne National Laboratory

DONG HUN LEE
Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY

TEAM-LEVEL MDRANGE POLICIES

- Provide multidimensional support for nested parallelism
  - Kokkos 4.0

ADDITIONS TO NESTED POLICIES

- MD versions of nested team execution policies
  - Supports multi dimension in nested parallel pattern

- TeamThreadRange
- **TeamThreadMDRange**
- TeamVectorRange
- **TeamVectorMDRange**
- ThreadVectorRange
- **ThreadVectorMDRange**

API FOR *MDRANGE

```
parallel_for(
  *MDRange<Rank<2>, TeamHandle>(team_handle, i0, i1),
  [=](int i, int j) { /* ... */ }
);
```

- Takes in `Rank<N, OuterDir, InnerDir>` that describes its iteration pattern
- Same behavior as regular `MDRangePolicy`
  - N is number of dimensions (required to be [2, 8])
  - `*Dir` are `enum class Iterate { Default, Left, Right }`
  - `Iterate` is used to choose iterating left-most dimension or right-most dimension
  - Only `OuterDir` is used for `*MDRange`

DEDUCTION GUIDES (CTAD) IN *MDRANGE

- Non ∗MDRange policies have functions for deduction purposes that return implementation-defined types

```
template <class TeamMemberType, class iType>
/* implementation defined */ TeamThreadRange(TeamMemberType team, iType count);
```

- In C++17 we can deduce Rank & TeamHandle from the constructor parameters

```
*MDRange(team, 4);  // NOT OK, violates i>=2

*MDRange(team, 4, 5);     // OK; *MDRange<Rank<2>, decltype(team)>
*MDRange(team, 4, 5, 6);  // OK; *MDRange<Rank<3>, decltype(team)>
*MDRange(team, 4, 5, 6, 2, 3, 4, 5, 6);  // OK, max num of extents allowed
```

TeamThreadMDRange

```
template <class Rank, typename TeamHandle>
class TeamThreadMDRange { /* ... */ };

TeamThreadMDRange(team, extent_1, extent_2, ...);
```

- Splits the index range `0` to *`extent`* over the threads of the team
  - *`extent`* is the backend-dependent rank that will be threaded

```
using TeamHandle = TeamPolicy<>::member_type;

parallel_for(TeamPolicy<>(N,AUTO),
  KOKKOS_LAMBDA(TeamHandle const& team) {

    int leagueRank = team.league_rank();

    auto range = TeamVectorMDRange(team, n0, n1, n2, n3);

    parallel_for(range,
      [=](int i0, int i1, int i2, int i3) {
        A(leagueRank, i0, i1, i2, i3) = B(leagueRank, i1) + C(i1, i2, i3);
    });
    team.team_barrier();

    int teamSum = 0;
    parallel_reduce(range,
        [=](int i0, int i1, int i2, int i3, int& vectorSum) {
          vectorSum += v(leagueRank, i, j, k, l);
        }, teamSum
    );
    single(PerTeam(team), [&leagueSum, teamSum]() { leagueSum += teamSum; });
    A_rowSum[leagueRank] = leagueSum;
  });
```

TeamVectorMDRange

```
template <class Rank, typename TeamHandle>
class TeamVectorMDRange { /* ... */ };

TeamVectorMDRange(team, extent_1, extent_2, ...);
```

- Splits an index range over the threads of the team and another index range over their vector lanes.
  - Ranks for threading and vectorization determined by the backend

```
using TeamHandle = TeamPolicy<>::member_type;

parallel_for(
    TeamPolicy<>(N, AUTO), KOKKOS_LAMBDA(TeamHandle const& team) {
        int leagueRank = team.league_rank();

        auto range = TeamVectorMDRange(team, n0, n1, n2, n3);

        parallel_for(range, [=](int i0, int i1, int i2, int i3) {
            A(leagueRank, i0, i1, i2, i3) = B(leagueRank, i1) + C(i1, i2, i3);
        });
        team.team_barrier();

        int teamSum = 0;
        parallel_reduce(
            range,
            [=](int i0, int i1, int i2, int i3, int& vectorSum) {
                vectorSum += v(leagueRank, i, j, k, l);
            },
            teamSum);
        single(PerTeam(team),
               [&leagueSum, teamSum]() { leagueSum += teamSum; });
        A_rowSum[leagueRank] = leagueSum;
    });
```

ThreadVectorMDRange

```
template <class Rank, typename TeamHandle>
class ThreadVectorMDRange { /* ... */ };

ThreadVectorMDRange(team, extent_1, extent_2, ...);
```

- Splits the index range `0` to *`extent`* over the vector lanes of the calling thread
  - *`extent`* is the backend-dependent rank that will be vectorized
  - Dispatched from a `TeamThreadRange` or `TeamThreadMDRange`

```
using TeamHandle = TeamPolicy<>::member_type;

parallel_for(
    TeamPolicy<>(N, Kokkos::AUTO), KOKKOS_LAMBDA(TeamHandle const& team) {
        int leagueRank = team.league_rank();

        auto teamThreadRange = TeamThreadRange(team, n0);
        auto threadVectorMDRange =
            ThreadVectorMDRange(team, n1, n2, n3);

        parallel_for(teamThreadRange, [=](int i0) {
            parallel_for(threadVectorMDRange, [=](int i1, int i2, int i3) {
                A(leagueRank, i0, i1, i2, i3) +=
                    B(leagueRank, i1) + C(i1, i2, i3);
            });
        });
        team.team_barrier();

        int teamSum = 0;
        parallel_for(teamThreadRange, [=, &teamSum](int const& i0) {
            int threadSum = 0;
            parallel_reduce(
                threadVectorMDRange,
                [=](int i1, int i2, int i3, int& vectorSum) {
                    vectorSum += D(leagueRank, i0, i1, i2, i3);
                },
                threadSum);

            teamSum += threadSum;
        });
    });
```

NESTED MDRANGE POLICIES

- Thread and Vector Parallelism:
  - Based on iteration direction (`OuterDir`)
    - Default direction computed from *`TeamHandle`*`::execution_space::array_layout`
  - For now, at most 2 dimensions are paralleled
    - Thread parallelism is applied to the slowest dimension
    - Vector parallelism is applied to the fastest dimension

- This was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative. Additionally, this research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.