OAK RIDGE National Laboratory

Experiences with Kokkos sorting in ArborX

Andrey Prokopenko

Oak Ridge National Laboratory (ORNL)

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

U.S. DEPARTMENT OF ENERGY

Sorting in ArborX

ArborX is a performance portable geometric search library.

Several uses of sort:

- Sorting primitives and queries along space-filling curve
- Sorting keys when constructing a dendrogram in HDBSCAN*
- Sorting dense cell indices in DBSCAN algorithm

Requirements:

- Most places require sort to get back **permutation**
- Often, there are many duplicates
- Sometimes we want to sort pairs

Sorting is critical! ArborX needs **sort_by_key** like functionality (otherwise, penalty).

More motivation…

Kokkos 4.2, CUDA 11.5, A100

Keys and values: View<unsigned*>. Keys are random, values initialized to iota.

Size : **30M | 500M**

- **Time [Kokkos::BinSort] : 0.026 | 0.474 [8x | 15x]**
  - BinSort followed by 2 x apply_permutation
- **Time [Kokkos::sort] : 0.088 | 2.062 [29x | 64x]**
  - Sort with custom comparison operator followed by apply_permutation
- **Time [Thrust] : 0.003 | 0.032**

Current status

```
template <typename ExecutionSpace, typename Keys, typename Values>
void sortByKey(ExecutionSpace const &space, Keys &keys, Values &values)
```

ArborX currently has sortByKey wrappers. It wraps:

- Nvidia: Thrust
- AMD: rocThrust
- SYCL: oneDPL
- Serial and OpenMP: Kokkos::BinSort

It is about 170 lines of code (see ArborX/src/kokkos_ext/ArborX_DetailsKokkosExtSort.hpp)

Things we currently do that we don't like

```
// Some versions of Clang fail to compile Thrust, failing with errors like
// this:
//    <snip>/thrust/system/cuda/detail/core/agent_launcher.h:557:11:
//    error: use of undeclared identifier 'va_printf'
// The exact combination of versions for Clang and Thrust (or CUDA) for this
// failure was not investigated, however even very recent version combination
// (Clang 10.0.0 and Cuda 10.0) demonstrated failure.
//
// Defining _CubLog here allows us to avoid that code path, however disabling
// some debugging diagnostics
//
// If _CubLog is already defined, we save it into ARBORX_CubLog_save, and
// restore it at the end
#    ifdef _CubLog
#      define ARBORX_CubLog_save _CubLog
#    endif
#    define _CubLog
#    include <thrust/device_ptr.h>
#    include <thrust/sort.h>
#    undef _CubLog
#    ifdef ARBORX_CubLog_save
#      define _CubLog ARBORX_CubLog_save
#      undef ARBORX_CubLog_save
#    endif
#  else // #if defined(KOKKOS_COMPILER_CLANG)
#    include <thrust/device_ptr.h>
#    include <thrust/sort.h>
#  endif // #if defined(KOKKOS_COMPILER_CLANG)
#endif   // #if defined(KOKKOS_ENABLE_CUDA)
```

```
#if ONEDPL_VERSION_MAJOR > 2022 ||                                             \
    (ONEDPL_VERSION_MAJOR == 2022 && ONEDPL_VERSION_MINOR >= 2)
  oneapi::dpl::sort_by_key(policy, keys.data(), keys.data() + n, values.data());
#else
  auto zipped_begin =
      oneapi::dpl::make_zip_iterator(keys.data(), values.data());
  oneapi::dpl::sort(
      policy, zipped_begin, zipped_begin + n,
      [](auto lhs, auto rhs) { return std::get<0>(lhs) < std::get<0>(rhs); });
#endif
}
```

```
#if defined(KOKKOS_ENABLE_CUDA)
#  if defined(KOKKOS_COMPILER_CLANG)

// Older Thrust (or CUB to be more precise) versions use __shfl instead of
// __shfl_sync for clang which was removed in PTX ISA version 6.4, also see
// https://github.com/NVIDIA/cub/pull/170.
#include <cub/version.cuh>
#if defined(CUB_VERSION) && (CUB_VERSION < 101100) && !defined(CUB_USE_COOPERATIVE_GROUPS)
#define CUB_USE_COOPERATIVE_GROUPS
#endif
```

What we would like to happen

Ideally, Kokkos would:

- Provide **sort_by_key**
- Provide **stable_sort_by_key** for reproducibility
- **Dispatch** them to Thrust, rocThrust and oneDPL (when available)
- (Possibly) provide native Kokkos Thrust like sorting (radix-based) (#5507?)
- (Stretch goal) Provide **segmented** and **semi** sorting

We can help with profiling and testing the functionality on our problems.

Questions?

https://github.com/aprokop/kokkos_perf

prokopenkoav@ornl.gov

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.