ArborX: a geometric search library

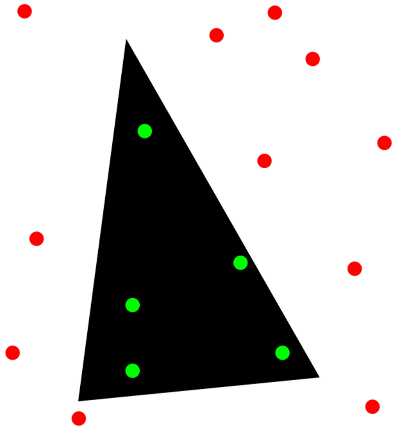
Andrey Prokopenko
Damien Lebrun-Grandié
Bruno Turcksin
Daniel Arndt

Oak Ridge National Laboratory (ORNL)

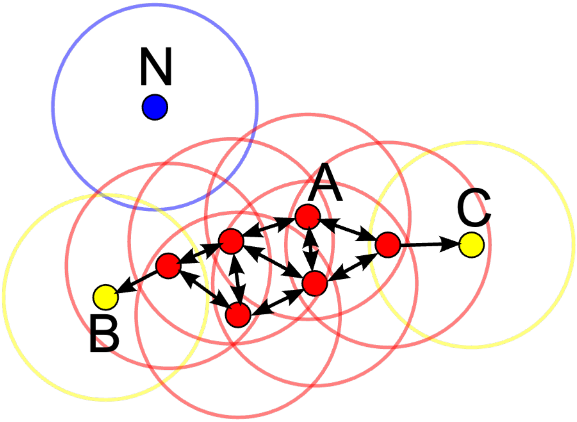
ORNL is managed by UT-Battelle LLC for the US Department of Energy

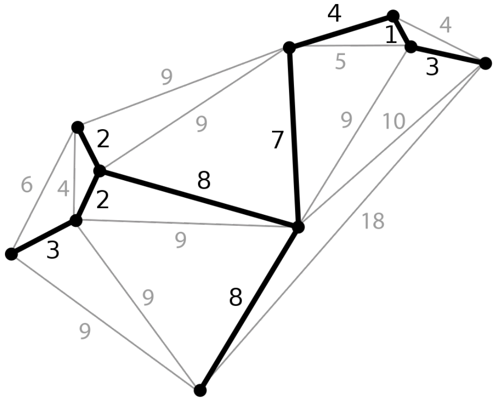
What is ArborX?

ArborX is an open-source **performance portable geometric search library** based on MPI+Kokkos.



- **Search**
  - k-nearest neighbors (k-NN)
  - Range search (radius search, intersections)
- **Ray Tracing**
- **Clustering algorithms**
  - Minimum spanning tree (Euclidean MST)
  - Density-based clustering (DBSCAN, HDBSCAN*)
- **Interpolation**
  - Moving Least Squares (MLS)





https://github.com/arborx/ArborX

Who is developing ArborX?

Core developer team

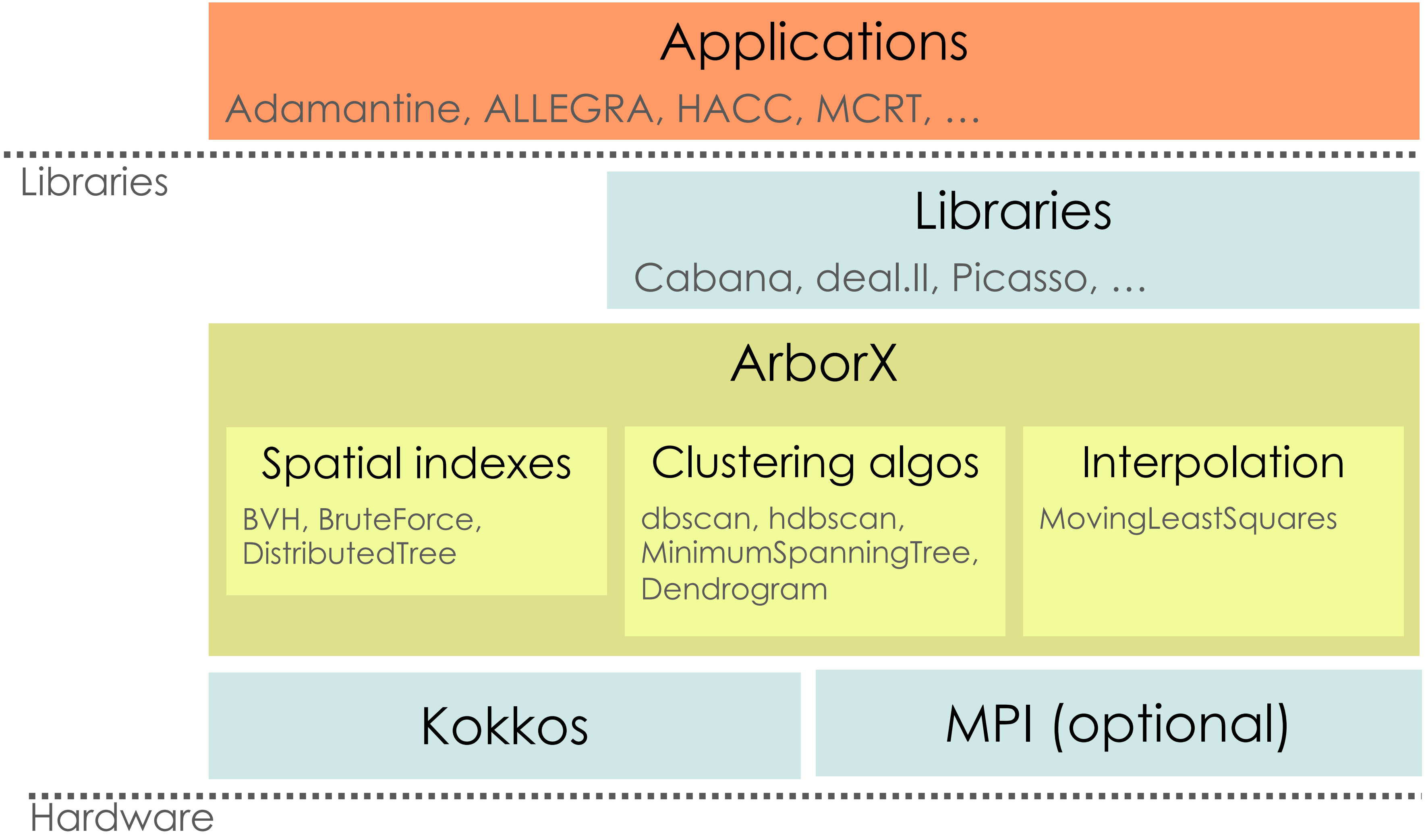
- Daniel Arndt*
- Damien Lebrun-Grandié*
- Andrey Prokopenko
- Bruno Turcksin*

Contributors

- Ana Gainaru
- Wenjun Ge
- Piyush Sao
- Yohann Bosqued

* Also Kokkos developers!

ArborX in the scientific software stack



Applications

Adamantine, ALLEGRA, HACC, MCRT, …

Libraries

Libraries

Cabana, deal.II, Picasso, …

ArborX

Spatial indexes

BVH, BruteForce, DistributedTree

Clustering algos

dbscan, hdbscan, MinimumSpanningTree, Dendrogram

Interpolation

MovingLeastSquares

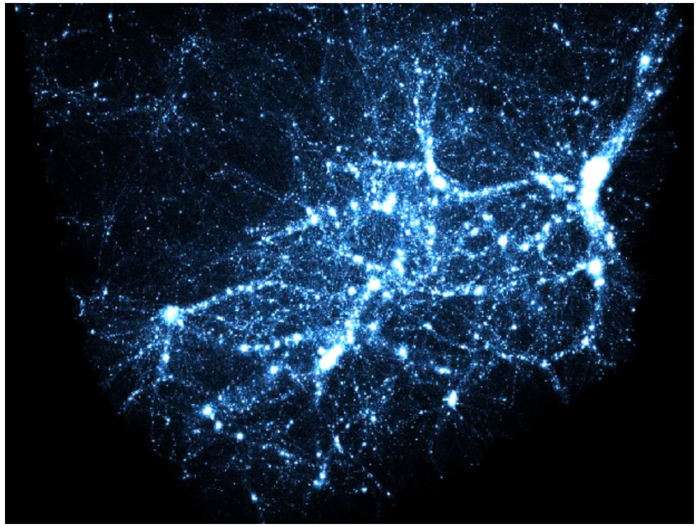
Kokkos

MPI (optional)

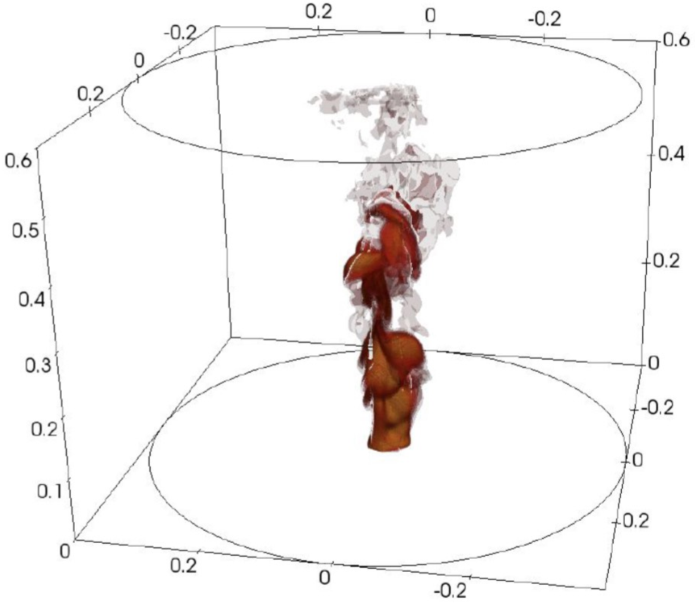
Hardware

Who uses ArborX?

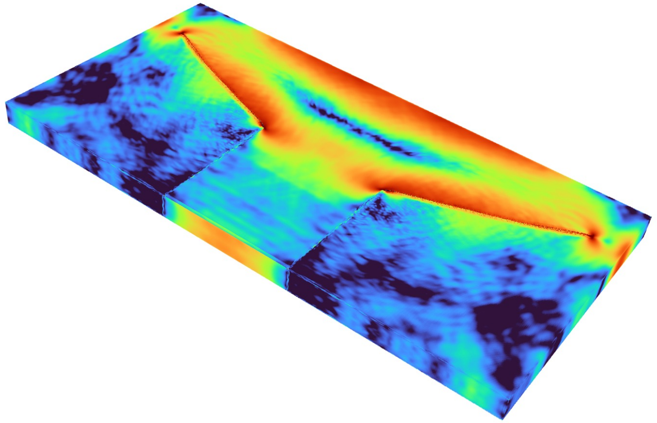
- **NimbleSM** contact mechanics
- **ALEGRA** shock hydrodynamics
- **LGRT** Lagrangian grid reconnection
- **deal.II** finite element library
- **DataTransferKit** solution transfer
- **MCRT** thermal radiation
- **Picasso** particle-in-cell
- **HACC/CosmoTools** clustering (dark matter)
- **Cabana** particle-based simulations
- **Adamantine** additive manufacturing
- …



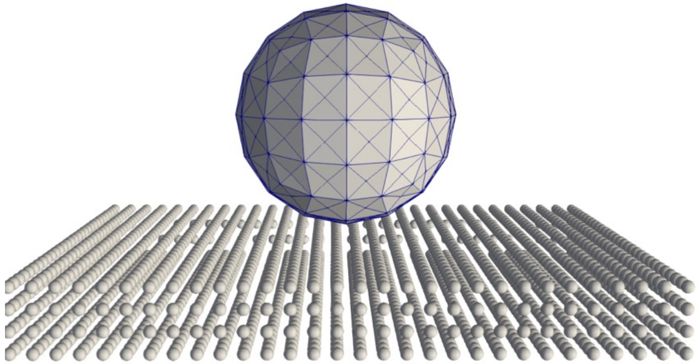
Cosmology (Credits: Nicholas Frontiere, ANL)



Combustion (Credits: Nicolas Tricard, UConn)



Additive manufacturing (Credits: Sam Reeve, ORNL)



Contact mechanics (Credits: Nicolas Morales, SNL)

Why Kokkos?

Context: start of US DOE Exascale Computing Project ~2017

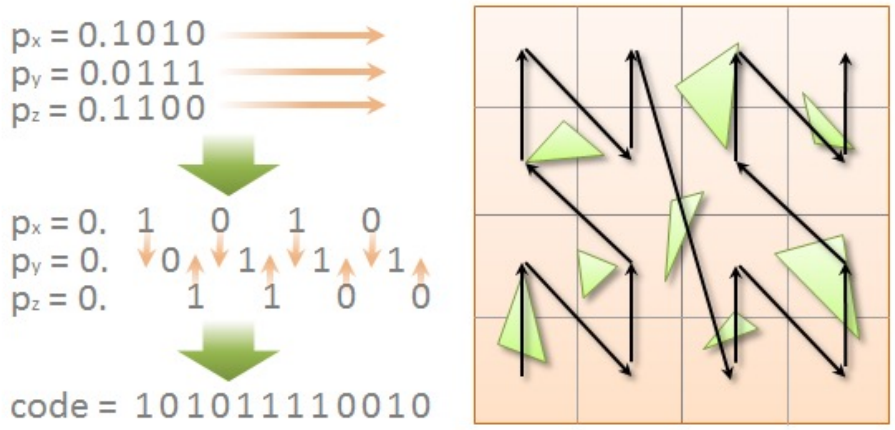
- Facing the unknown beyond Summit
- SYCL not around at that time
- RAJA? Kokkos? Roll our own?

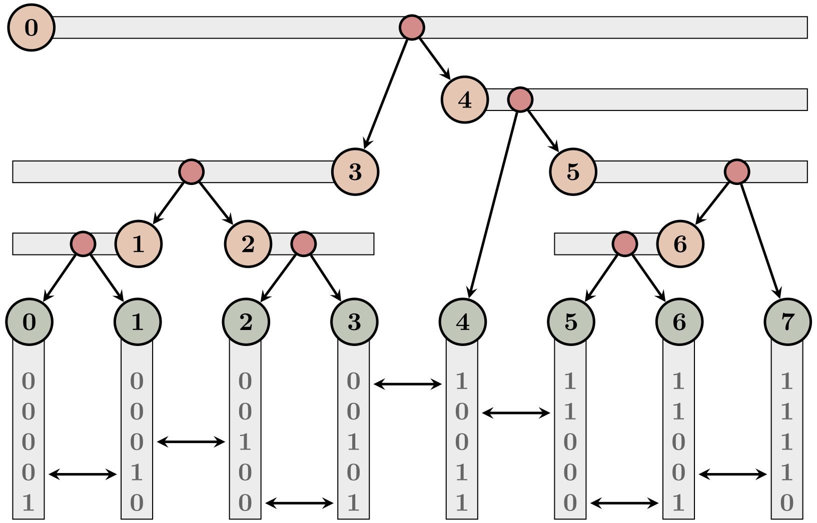
Join forces with Kokkos

- More than a programming model
- Ecosystem with debugging and profiling tools, math libraries, etc.
- Building a community

The workhorse: Bounding Volume Hierarchy (BVH)

- Impose order in which leaf nodes appear in the tree (Z-curve/Morton codes)
- Each internal node is a linear range over leaf nodes
- The splits are determined according to the highest bit that differs between the Morton codes within the given range
- Can be constructed fully in parallel

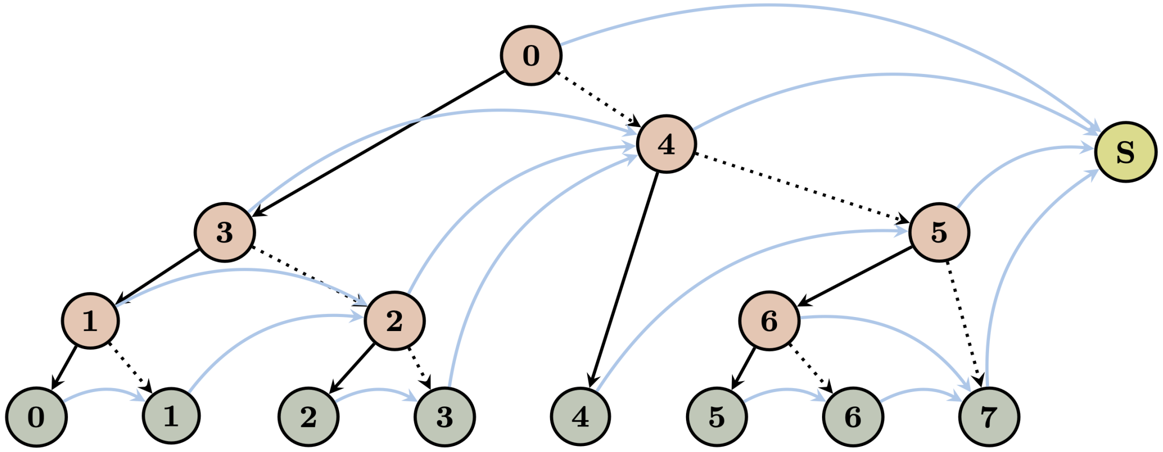




Creating the index

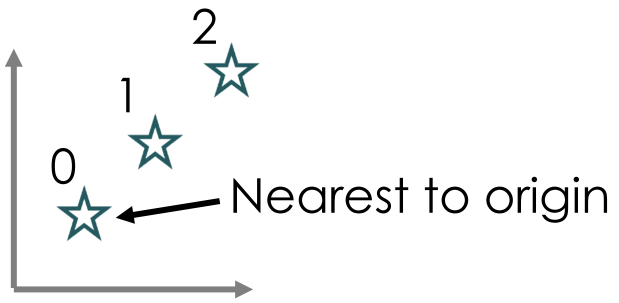
- Building the data structure from a collection of boundable geometric objects O(N log N)
- Tree structure
  - N leaf and N-1 internal nodes
  - Store bounding volume, left child, and “rope”
  - Implementation detail not exposed in the API
- Interchangeable with other data structures provided (BruteForce)
- Distributed tree also uses MPI_Comm

```
ArborX::BoundingVolumeHierarchy<MemorySpace>::
BoundingVolumeHierarchy

BoundingVolumeHierarchy() noexcept; // (1)

template <typename ExecutionSpace, typename Primitives>
BoundingVolumeHierarchy(ExecutionSpace const &space,
              Primitives const &primitives); // (2)
```



“Hello, World!” program with ArborX

Nearest neighbor search



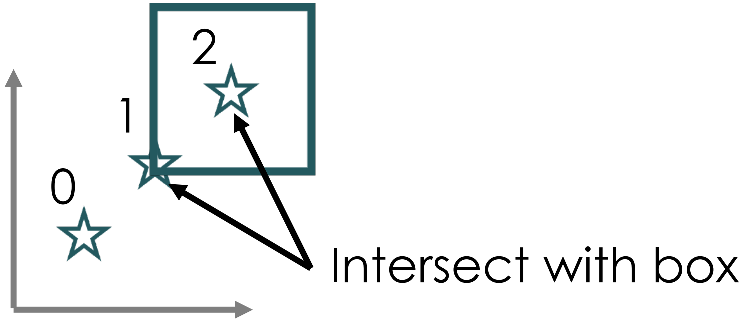
```
#include <ArborX.hpp>
#include <Kokkos_Core.hpp>
int main(int argc, char *argv[])
{
 Kokkos::initialize(argc, argv);
 {
  Kokkos::DefaultExecutionSpace exec;
  ArborX::BoundingVolumeHierarchy bvh(
    exec, to-view(
          {
            {1., 1., 1.}, // 0
            {2., 2., 2.}, // 1
            {3., 3., 3.}, // 2
          }));
  bvh.query(
    exec, to-view(ArborX::nearest(ArborX::Point{0., 0., 0.})),
    KOKKOS_LAMBDA(auto /*predicate*/, int primitive_index) {
     printf("Nearest to origin is %d\n", primitive_index);
    });
 }
 Kokkos::finalize();
 return 0;
}
// Prints "Nearest to origin is 0"
```

build data structure

search

“Hello, World!” program with ArborX

Intersection with geometries



Intersect with box

Unchanged

```
#include <ArborX.hpp>
#include <Kokkos_Core.hpp>
int main(int argc, char *argv[])
{
 Kokkos::initialize(argc, argv);
 {
  Kokkos::DefaultExecutionSpace exec;
  ArborX::BoundingVolumeHierarchy bvh(
    exec, to-view(
          {
            {1., 1., 1.}, // 0
            {2., 2., 2.}, // 1
            {3., 3., 3.}, // 2
          }));
  bvh.query(exec, to-view(ArborX::intersects(
         ArborX::Box{{2., 2., 2.}, {4., 4., 4.}})),
    KOKKOS_LAMBDA(auto /*predicate*/, int primitive_index) {
     printf(”Found %d\n", primitive_index);
    });
 }
 Kokkos::finalize();
 return 0;
}
```

// Prints ”Found 1\nFound 2\n” or ”Found 2\nFound 1\n”

Open slide master to edit

Access traits

- Customization point
- Opt-in mechanism to tell ArborX
  - where does the data reside
  - how much of it
  - how to access
- Allowed to specialize ArborX::AccessTraits class template for user-defined type
- Available both for "primitives" and "predicates"

```
struct PointCloud
{
  float *d_x;
  float *d_y;
  float *d_z;
  int N;
};
```

Some user-defined type with coordinates allocated using cudaMalloc()

```
template <>
struct ArborX::AccessTraits<PointCloud, ArborX::PrimitivesTag>
{
 using memory_space = Kokkos::CudaSpace;

  static KOKKOS_FUNCTION size_t size(PointCloud const &cloud)
  {
    return cloud.N;
  }
  static KOKKOS_FUNCTION
  auto get(PointCloud const &cloud, size_t i)
  {
    return ArborX::Point{cloud.d_x[i], cloud.d_y[i], cloud.d_z[i]};
  }
};
```

Allocated in CUDA device memory

Returns number of primitives

Access specified primitive

Queries and callbacks

- Callbacks are another customization point
- Specify what to do when primitives meet a predicate
- Able to store results in compressed sparse row format

```
struct PrintfCallback
{
 template <typename Predicate, typename OutputFunctor>
 KOKKOS_FUNCTION void operator()(Predicate,
        int primitive_index, OutputFunctor const &out) const
 {
  printf("Found %d from functor\n", primitive_index);
  out(primitive_index);
 }
};
```

```
ArborX::BoundingVolumeHierarchy<MemorySpace>::query

template <typename ExecutionSpace, typename Predicates,
          typename Callback>
void query(ExecutionSpace const& space,
      Predicates const& predicates,
      Callback const& callback) const; // (1)

template <typename ExecutionSpace, typename Predicates,
          typename Indices, typename Offsets>
void query(ExecutionSpace const& space,
      Predicates const& predicates,
      Indices& indices,
      Offsets& offsets) const; // (2)

template <typename ExecutionSpace, typename Predicates,
          typename Callback, typename Values, typename Offsets>
void query(ExecutionSpace const& space,
      Predicates const& predicates,
      Callback const& callback,
      Values& values,
      Offsets& offsets) const; // (3)
```

What ArborX uses from Kokkos

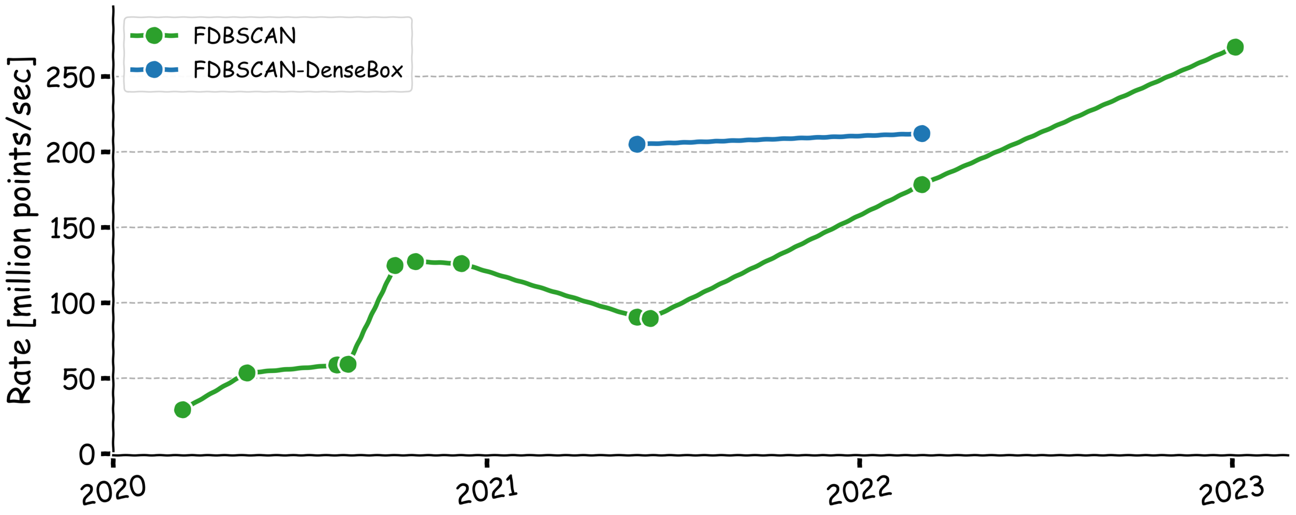
- Mostly **parallel_{for,reduce,scan}** with regular **RangePolicy**
- Few **TeamPolicy**
- **View**s, **Array**s, **pair**
- Synchronization primitives (atomic functions, memory fences)
- A few **algorithms** (most importantly sorting)
- Detection idioms (**Kokkos::is_detected**)
- Mathy things (finite_min/max, abs, pow, sqrt, bit_cast)
- Profiling hooks (diligently annotating regions, kernels, and allocations)
- Custom reductions

How ArborX extends Kokkos (we’d like not to)

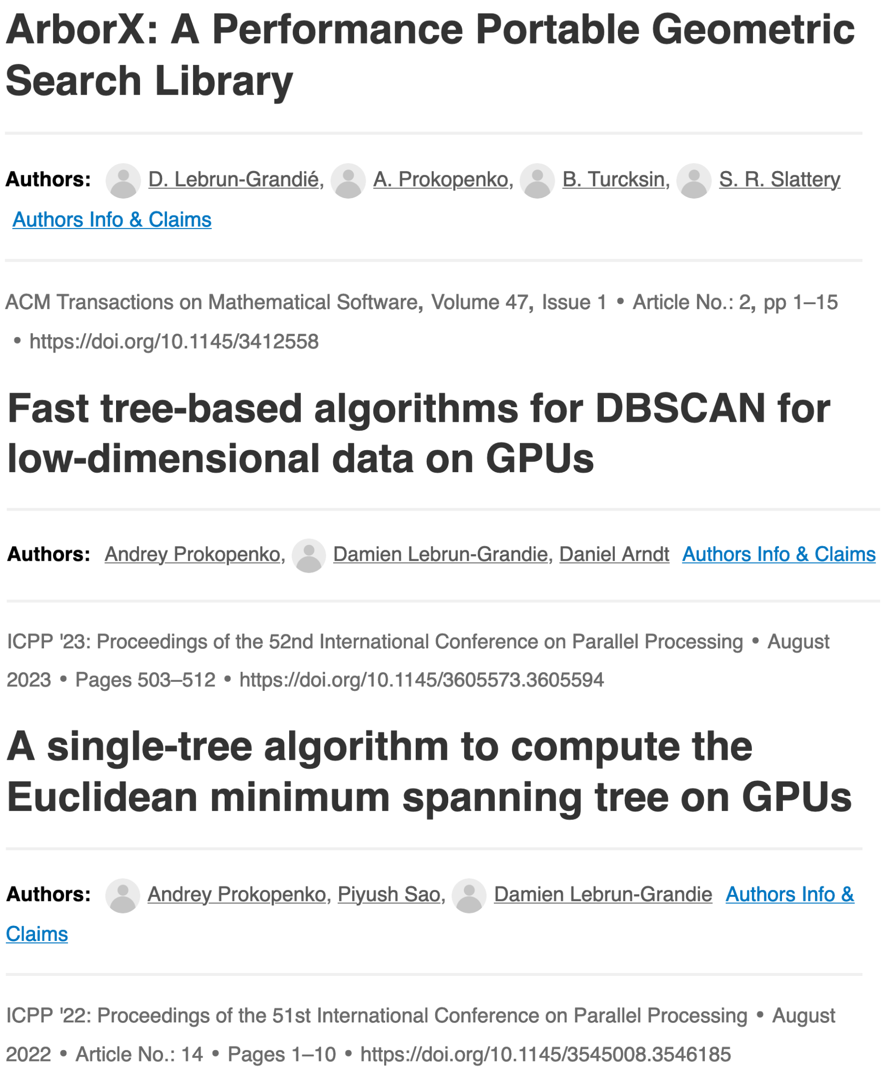
- **is_accessible_from[_host]**
- **sortByKey()**
- **swap()**
- **version()**
- **lastElement**
- **reallocWithoutInitializing**
- **clone, cloneWithoutInitializingNorCopying**
- **create_layout_right_mirror_view_no_init, create_layout_right_mirror_view_and_copy**
- **exclusivePrefixSum**
- **min(View), max(View), accumulate(View), adjacentDifference(View)**
  - We haven’t played with std algorithms in Kokkos, hoping to see some performance results first

How fast is ArborX?

- HACC 37M: 0.26s (Nvidia A100), 0.41s (AMD MI250x GCD).
- 1B 3D particles in 7.3s (1s construction, 6.3s query + clustering) on a single A100
- Constantly improving performance



State-of-the-art implementations



ArborX: A Performance Portable Geometric Search Library

Authors: D. Lebrun-Grandié, A. Prokopenko, B. Turcksin, S. R. Slattery Authors Info & Claims

ACM Transactions on Mathematical Software, Volume 47, Issue 1 • Article No.: 2, pp 1–15 • https://doi.org/10.1145/3412558

Fast tree-based algorithms for DBSCAN for low-dimensional data on GPUs

Authors: Andrey Prokopenko, Damien Lebrun-Grandie, Daniel Arndt Authors Info & Claims

ICPP '23: Proceedings of the 52nd International Conference on Parallel Processing • August 2023 • Pages 503–512 • https://doi.org/10.1145/3605573.3605594

A single-tree algorithm to compute the Euclidean minimum spanning tree on GPUs

Authors: Andrey Prokopenko, Piyush Sao, Damien Lebrun-Grandie Authors Info & Claims

ICPP '22: Proceedings of the 51st International Conference on Parallel Processing • August 2022 • Article No.: 14 • Pages 1–10 • https://doi.org/10.1145/3545008.3546185

Is ArborX really “single-source”?

- ArborX achieves good performance from single-source Kokkos code
- From our experience
  - Often, optimization led to specialization of data structures and algorithms
  - But, as code matured, it typically converged towards a single implementation

Few exceptions

- Returning a value instead of a reference with HIP in a bounding volume accessor on the device side because compiler had trouble generating efficient vector instructions
- Recompute distances on GPU or store them on thread local stack in nearest traversals
- Using slightly different “flavor” of the algorithm for the serial implementation of the minimum spanning tree and union-find algorithms

What does the future hold?

- New interface (API v2) to provide more flexibility
- Finishing touches for HDBSCAN* and interpolation algorithms
- RTX hardware support (e.g., OptiX)
- Multi-dimensional (>10) search
- Approximate search

If you have an interesting problem,

or simply want to learn more,

talk to us!!!

Questions?

https://github.com/arborx/ArborX

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.