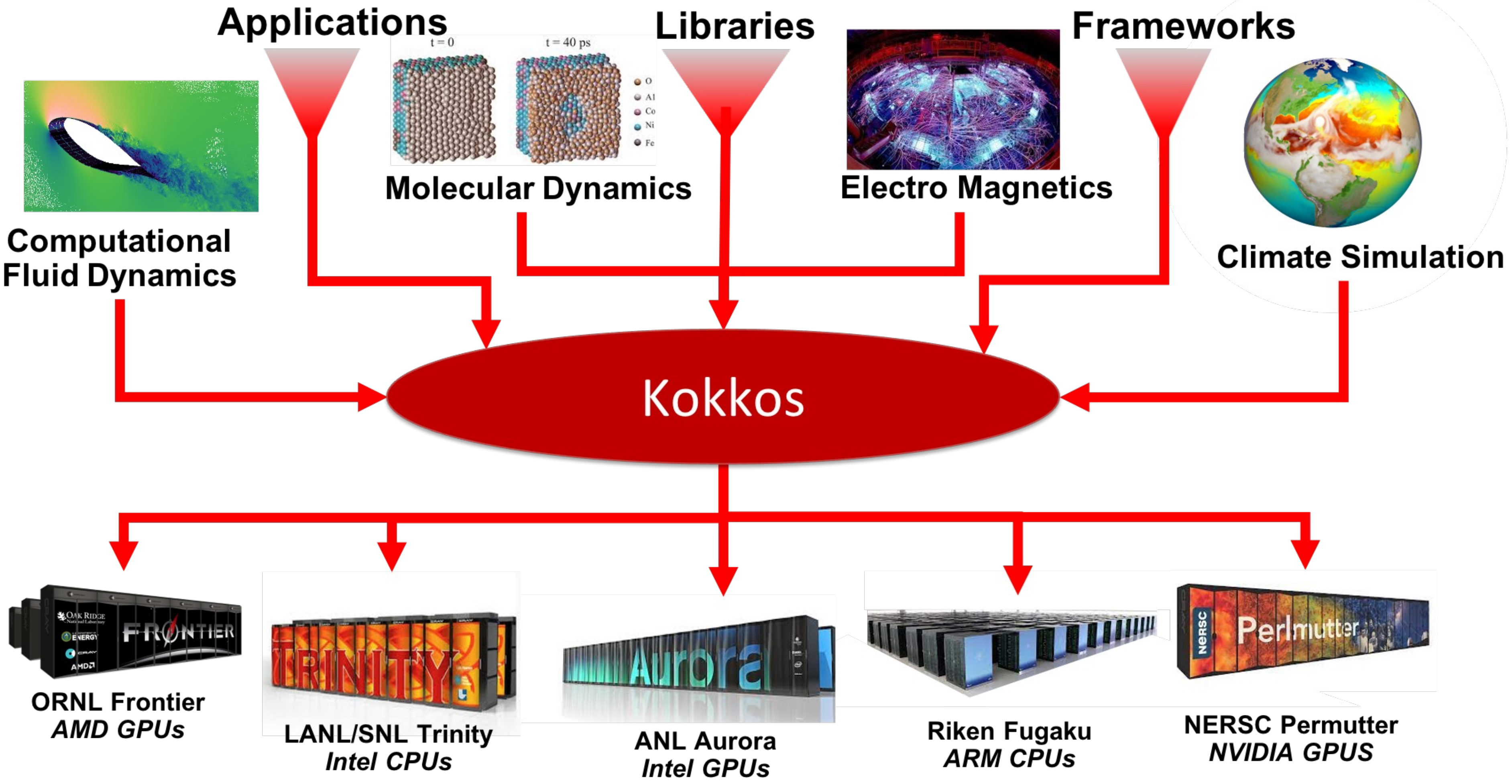
Overcoming Today’s Limitations of Standard C++ with Kokkos

Damien Lebrun-Grandié

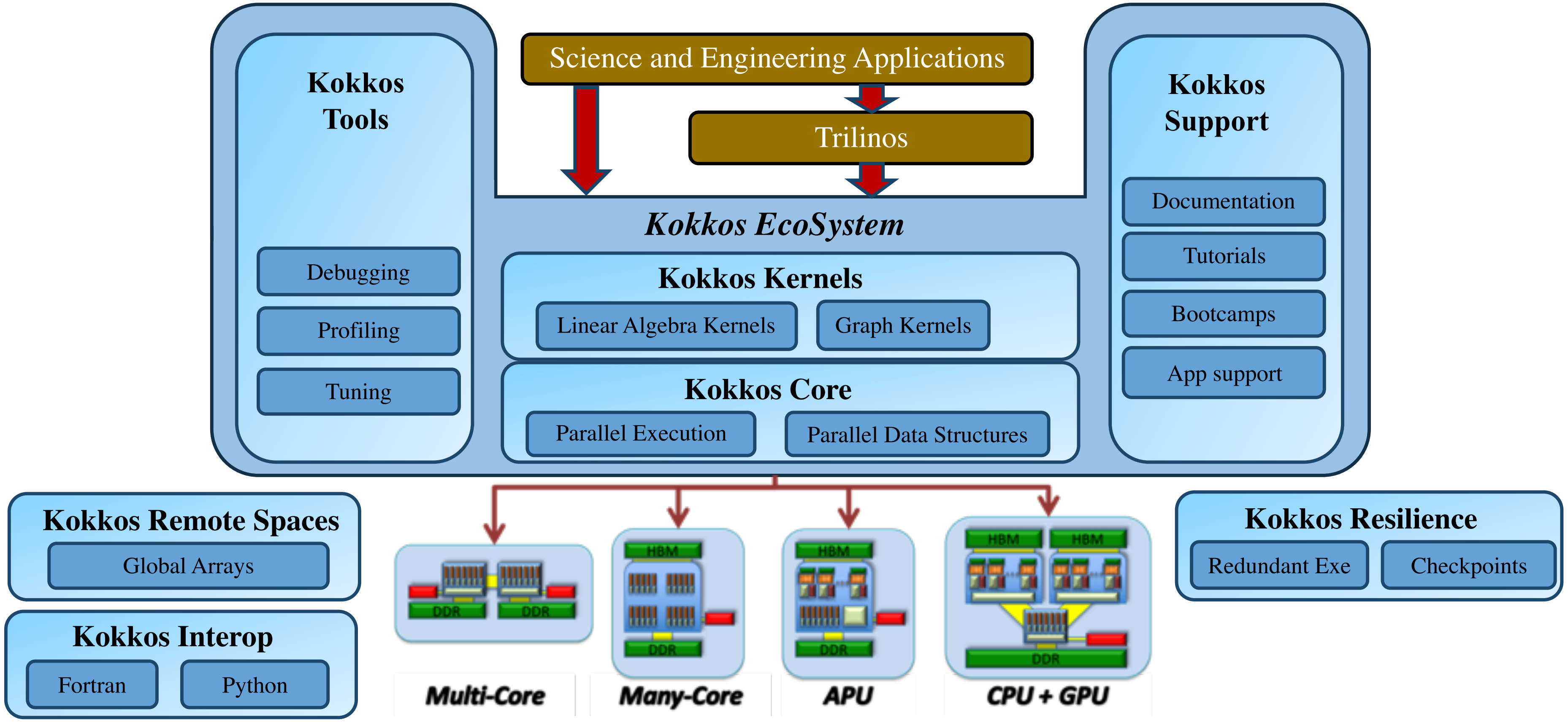
Content

- Brief overview of Kokkos
- Memory and execution spaces
- Hierarchical parallelism



Applications
Libraries
Frameworks
t = 0
t = 40 ps
O
Al
Co
Ni
Fe
Computational Fluid Dynamics
Molecular Dynamics
Electro Magnetics
Climate Simulation
Kokkos
FRONTIER
TRINITY
Aurora
Perlmutter
ORNL Frontier
AMD GPUs
LANL/SNL Trinity
Intel CPUs
ANL Aurora
Intel GPUs
Riken Fugaku
ARM CPUs
NERSC Permutter
NVIDIA GPUS

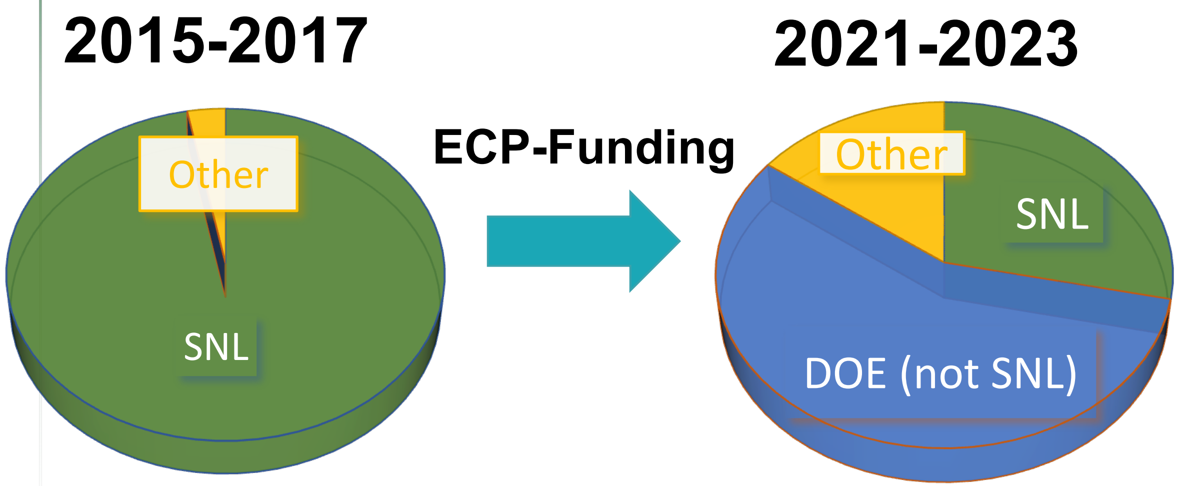
The Kokkos ecosystem



The Kokkos team

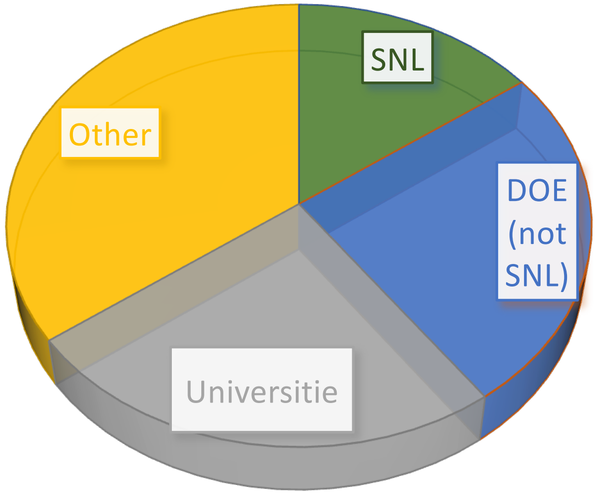
Kokkos Core – Contributions and Usage

Contributions



Usage

https://kokkosteam.slack.com
>1000 Registered Users
>130 Institutions



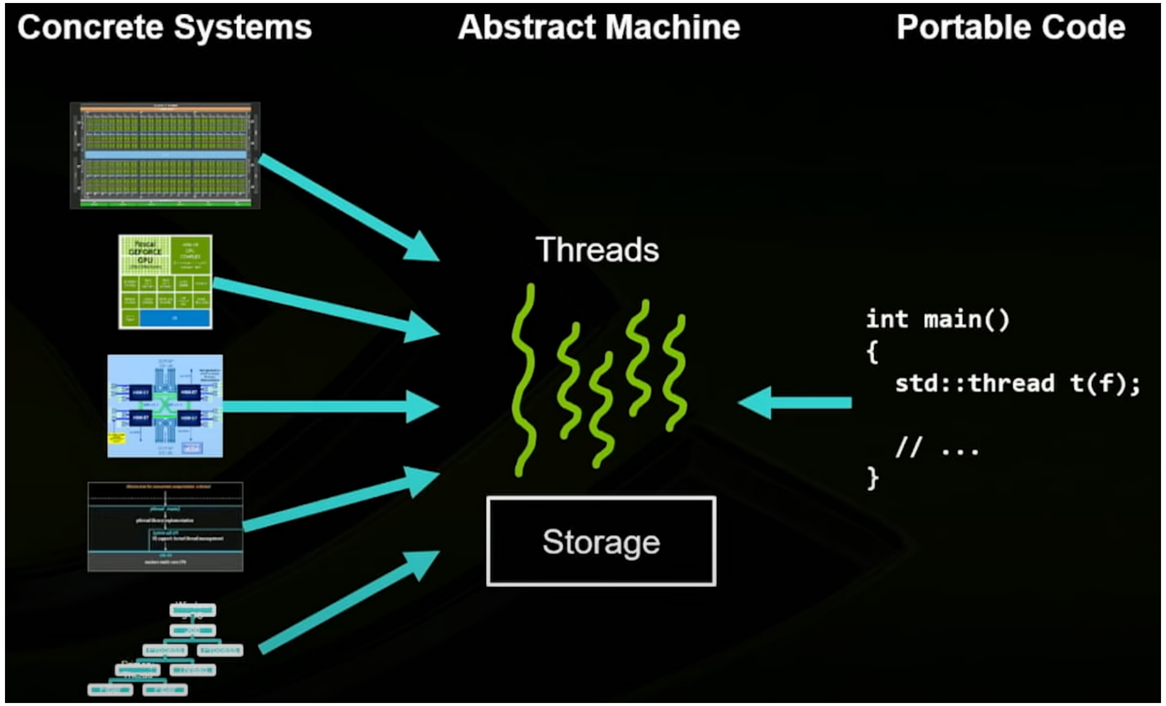
What do we mean by “Standard C++”

- Code you can write today **without** using **language extensions** or **additional libraries**, which is portable to other compiler and systems and can be ”automatically” accelerated with GPUs.
- Which excludes
  - CUDA/HIP require use of __host__ and __device__ attributes on functions and triple chevron syntax <<<…>>> for GPU kernel launches
  - OpenACC/OpenMP use **#pragma** directives to control GPU accelerations
  - Thrust or oneDPL let you express parallelism portably but only support a limited number of CPU and GPU backends

Kokkos execution and memory spaces

Standard C++ abstract machine

- **Threads of execution** evaluating functions that operate on objects that are in a **flat storage** space
- But…
- No notion of hierarchy (caches, etc.)
- **No** concept of **host or device** memory, nor accessibility



Adelstein Lelbach, CppCon 2018

Revisions of the C++ standard

Enable accelerating with GPUs

Standard way to take advantage of multicore processors

C++11

Memory model
Concurrent execution model
Concurrency support library

C++14

Atomic extensions

C++17

Parallel algorithms

C++20

C++23

mdspan

C++98/03

Dynamic memory management
Containers library
Algorithms library

GPU-enabled implementations today

NVIDIA HPC SDK

- Offloading of parallel algorithms to NVIDIA GPUs
- Enabled with the `-stdpar[=gpu]` option to NVC++
- Relies on CUDA Unified Memory for all data movement between CPU and GPU memory
- Automatically migrating data towards the processor using it

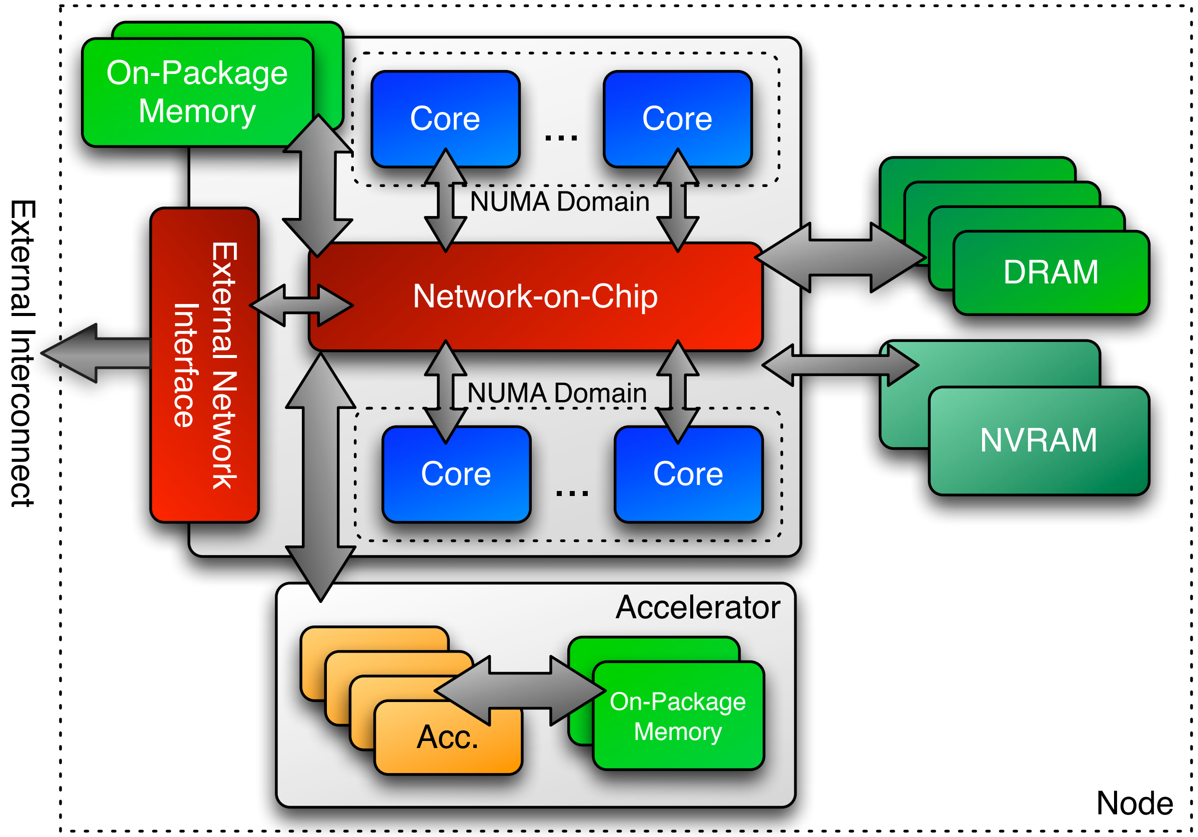
```
std::for_each(
    std::execution::par_unseq,
    v.begin(), v.end(),
    [](int& x) { x = x * x; });
```

Intel OneAPI DPC++

- Support for Intel, NVIDIA, and AMD GPU devices using oneDPL device execution policies (non-standard)
  std::par and std::par_unseq run on the host via TBB or OpenMP backend
- Unified Shared Memory
  C-style memory management via sycl::malloc_{device,host,shared}
- Or SYCL buffer objects to pass data to device (oneAPI runtime controls data movement)

```
std::fill(
    oneapi::dpl::execution::make_device_policy(queue),
    begin(v), end(v), 42);
```

Target machine for Kokkos programming model

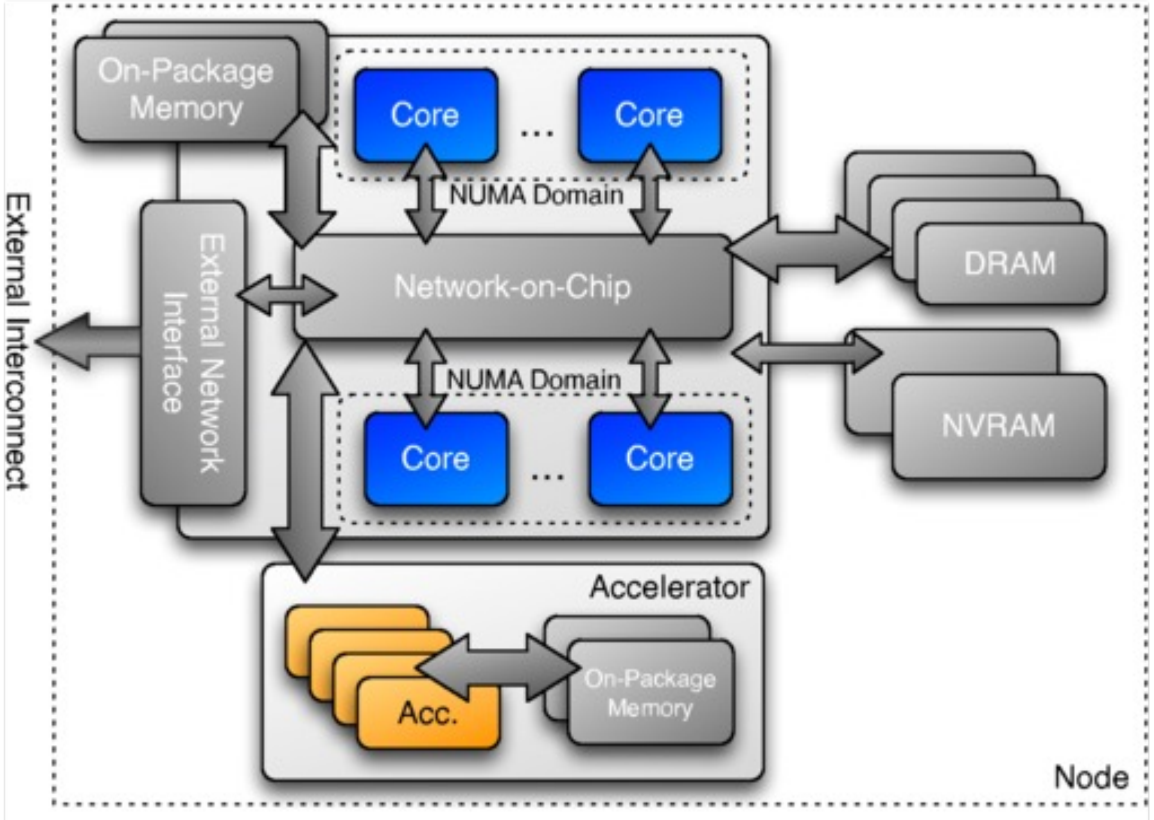


Kokkos **execution** spaces

- Define where kernels get executed and what backend to use

  (e.g. Serial, Threads, OpenMP, Cuda, HIP, …)
- Execution space instances encapsulating CUDA/HIP stream or SYCL queue

Always available:
DefaultExecutionSpace
DefaultHostExecutionSpace

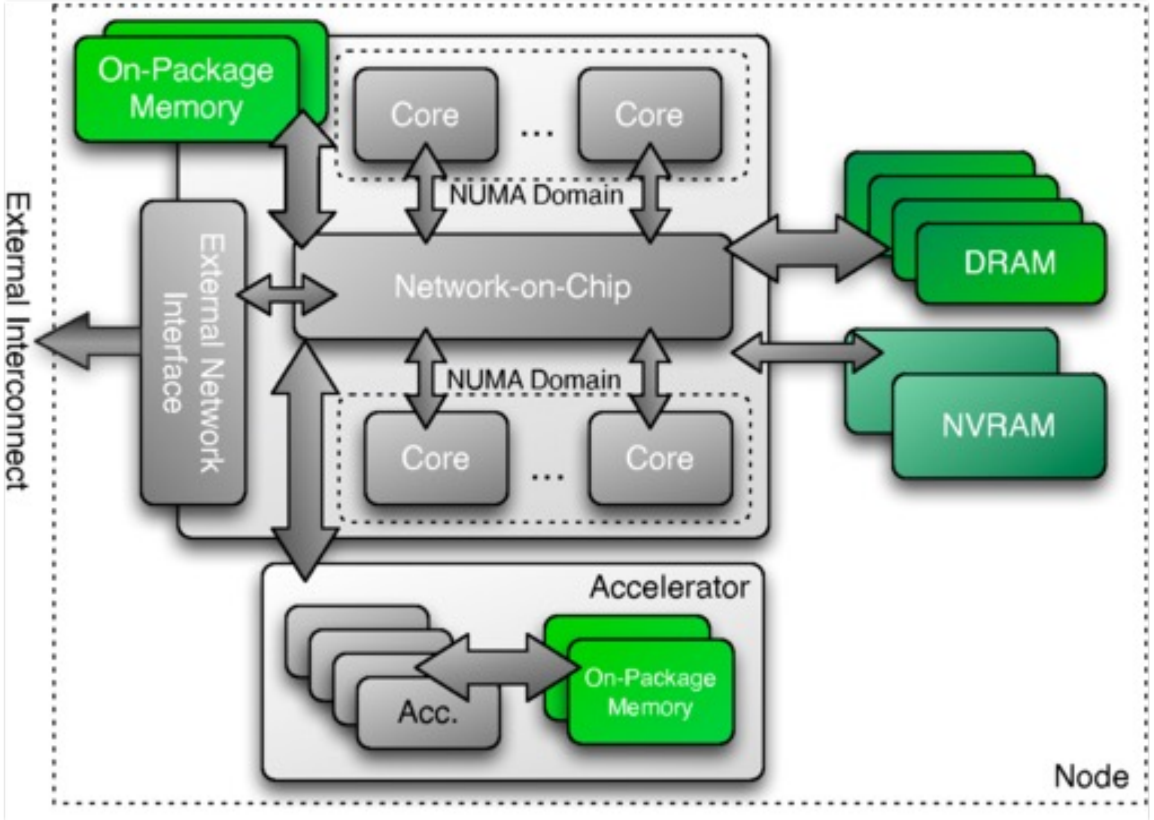


Kokkos **memory** spaces

- Define “where” and “how” memory allocation and access take place

  (e.g. HostSpace, SYCLDeviceUSMSpace, SYCLSharedUSMSpace, SYCLHostUSMSpace)

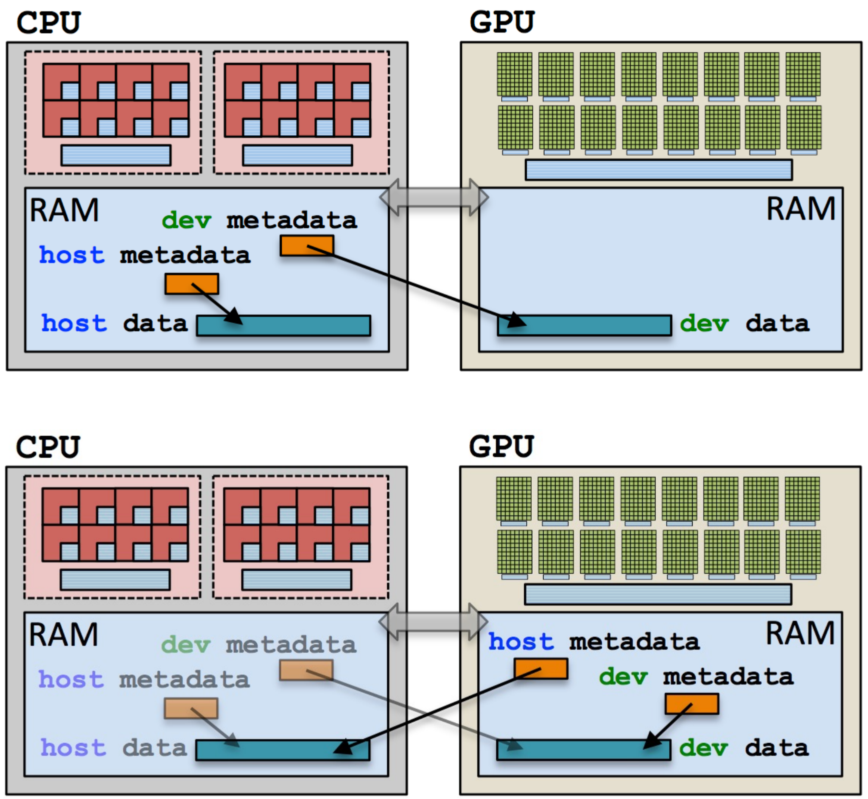
Always available:
HostSpace
SharedSpace
SharedHostPinnedSpace
ExecutionSpace::memory_space
ExecutionSpace::scratch_memory_space



Accessibility of allocations from target devices

memory access violation?

```
Kokkos::parallel_for(
   "Fill", N, KOKKOS_LAMBDA(int i) { v(i) = value; })
```



- Kokkos cannot introspect user-provided functors
- But it provides a facility to check accessibility and catch most bugs at compile-time rather than runtime

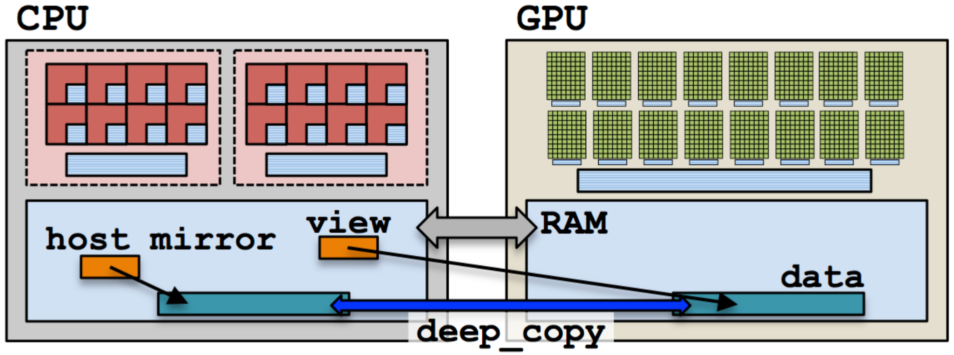
```
static_assert(is_accessible_from<
   typename View::memory_space,
   DefaultExecutionSpace>::value);
```

Kokkos user remains in control of data placement

Explicit data movement

```
auto v_host = create_mirror_view_and_copy(HostSpace(), v);
```

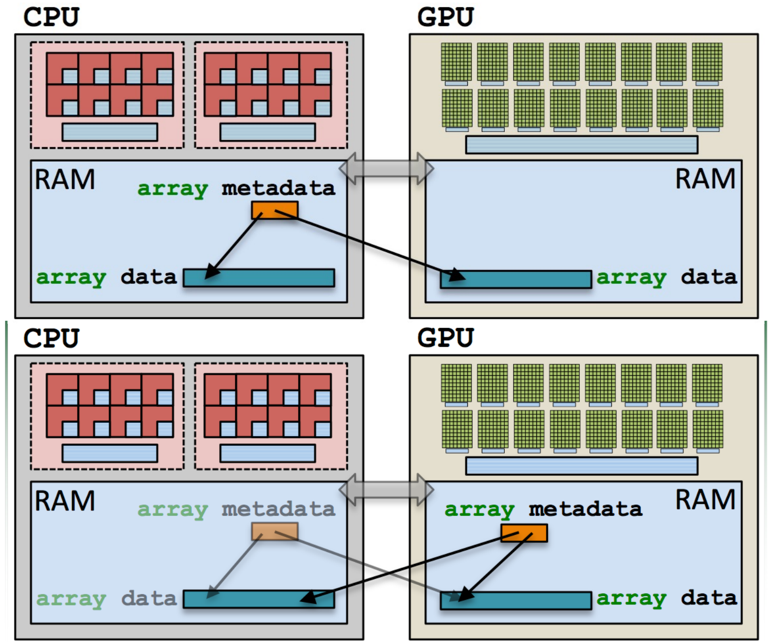
- Explicit data transfer between host and device
- no-op when view is already accessible from the CPU



Managed memory

- Managed memory accessible from all CPUs and GPUs in the system as a single, coherent memory image with a common address space

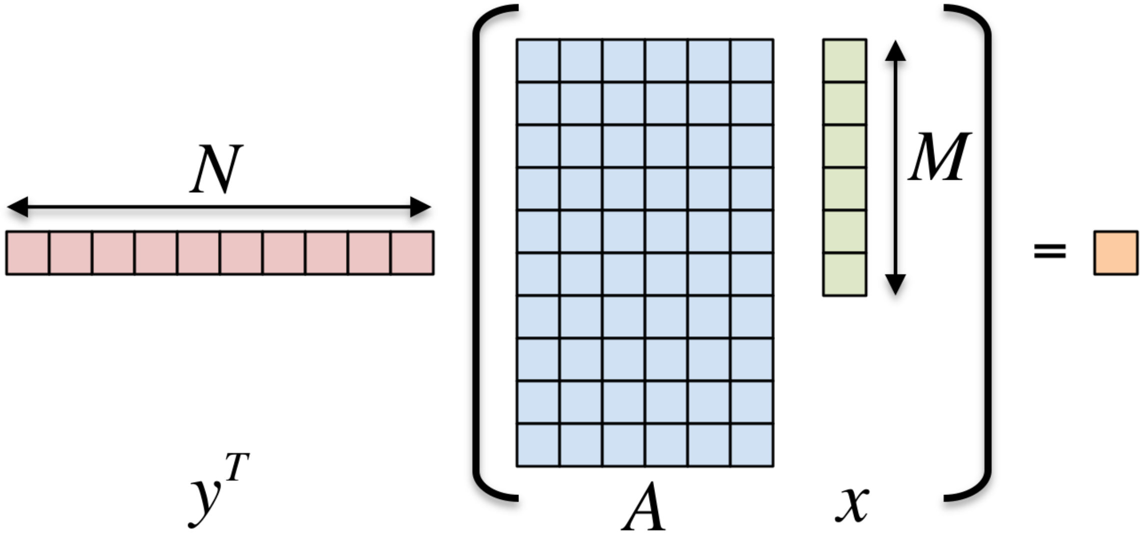
```
View<float*, SharedSpace> w("w", N);
```



Hierarchical parallelism

Example: inner product <y, A*x>

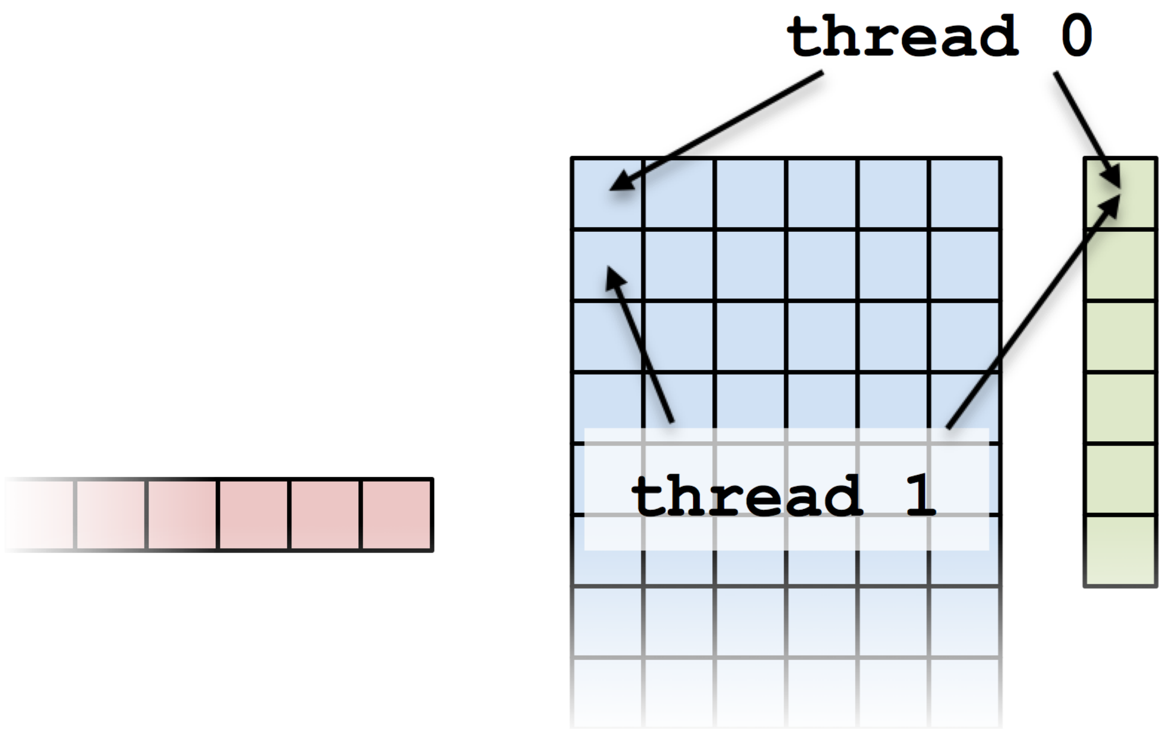
```
double result = 0.;
for (int i = 0; i < N, ++i) {
  double Ax_i = 0.;
  for (int j = 0; j < M, ++j) {
    Ax_i += A(i, j) * x(j);
  }
  result += y(i) * Ax_i;
}
```



How to parallelize using C++17 parallel algorithms?

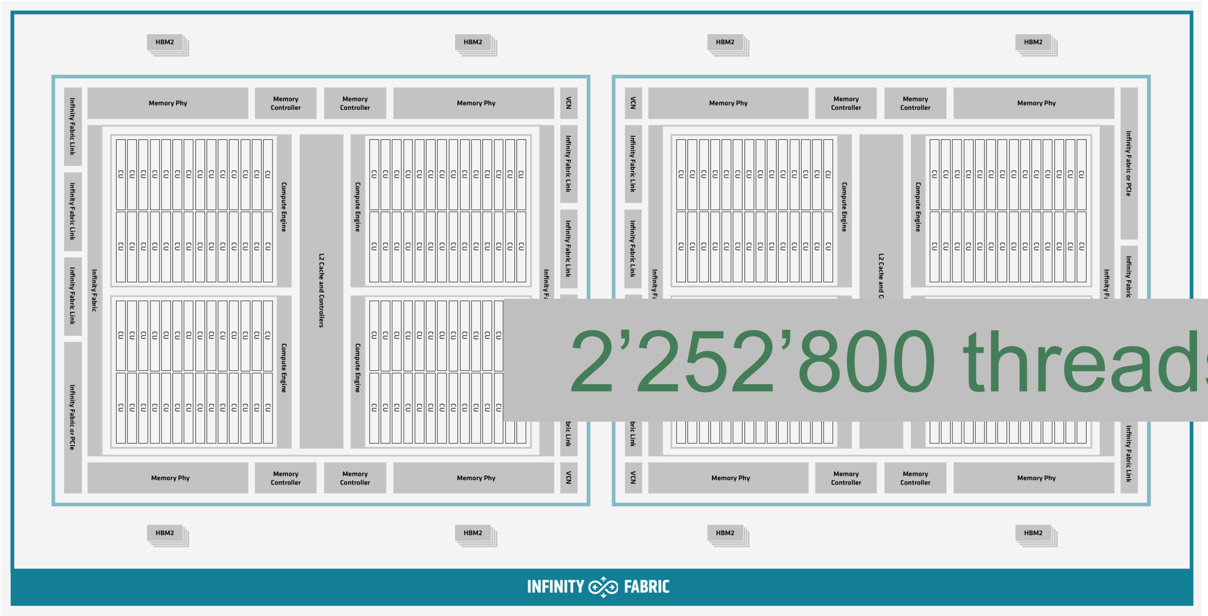
Accelerate <y,A*x> with standard parallelism

```
double result = std::reduce(
   std::execution::par_unseq,
   counting_iterator(0), counting_iterator(N),
   0.,
   [=](int i) {
      double Ax_i = 0.;
      for (int j = 0; j < M; ++j) {
        Ax_i += A(i, j) * x(j);
      }
      return y(i) * Ax_i;
   });
```



Problem: What if we don't have enough rows to saturate the GPU?

Frontier compute node

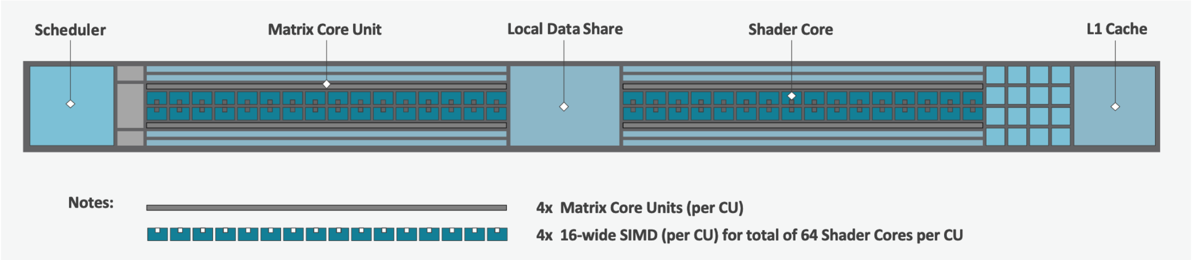


2’252’800 threads

[1x] 64-core AMD “Optimized 3rd Gen EPYC” CPU
[4x] MI250x each with 2 GCDs
Each GCD contains 110 CUs
64 GB of HBM accessible at 1.6 TB/s

4 CEs which dispatch wavefronts to CUs
vavefronts from a single workgroup are
gned to the same CU

Work items in a wavefront are scheduled in units of 64 called wavefronts
Up to 64 KB of LDS can be allocated



Each CU has 4 MCUs and 4 16-wide SIMD units
Each wavefront is assigned to a single 16-wide SIMD unit

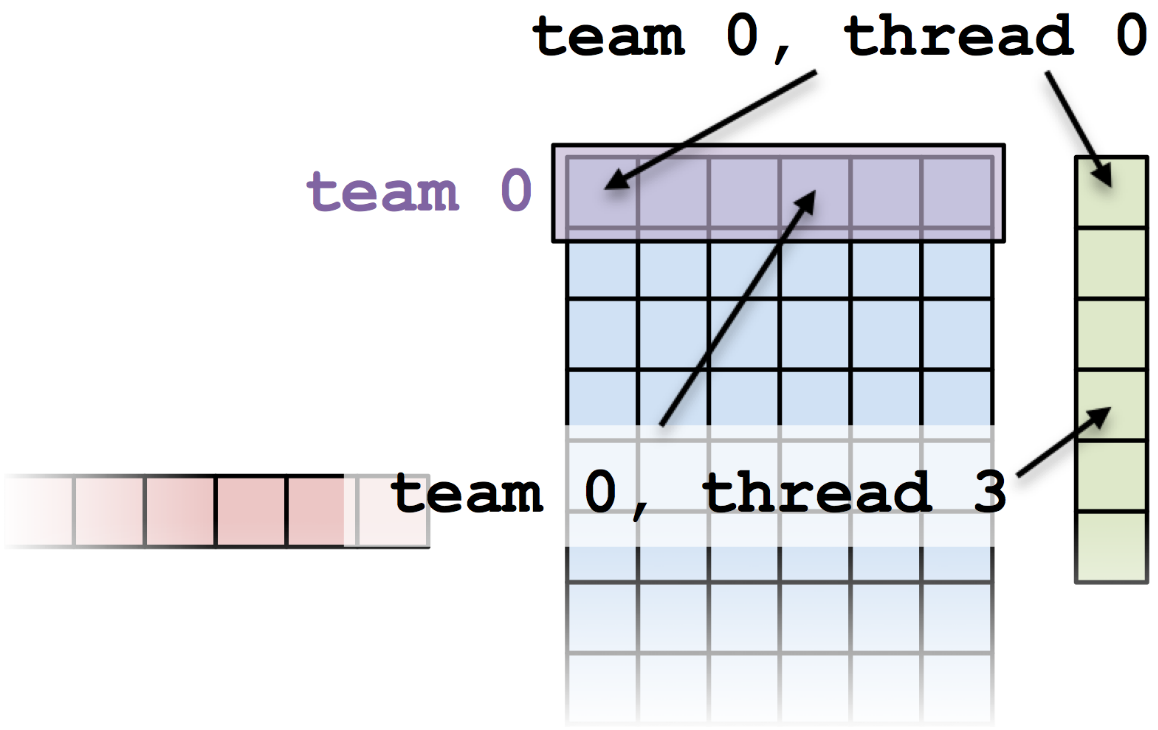
Each CU maintains an instruction buffer for 10 wavefronts

Hierarchical parallelism in Kokkos

- Exploit multiple level of shared-memory parallelism
  These levels include thread teams, threads within a team, and vector lanes.
- Able to nest these levels of parallelism, and execute parallel_for(), parallel_reduce(), or parallel_scan() at each level
- Syntax differs only by the execution policy which is the 1st argument to the parallel_*
- Also exposing a "scratch pad" memory which provides thread private and team private allocations

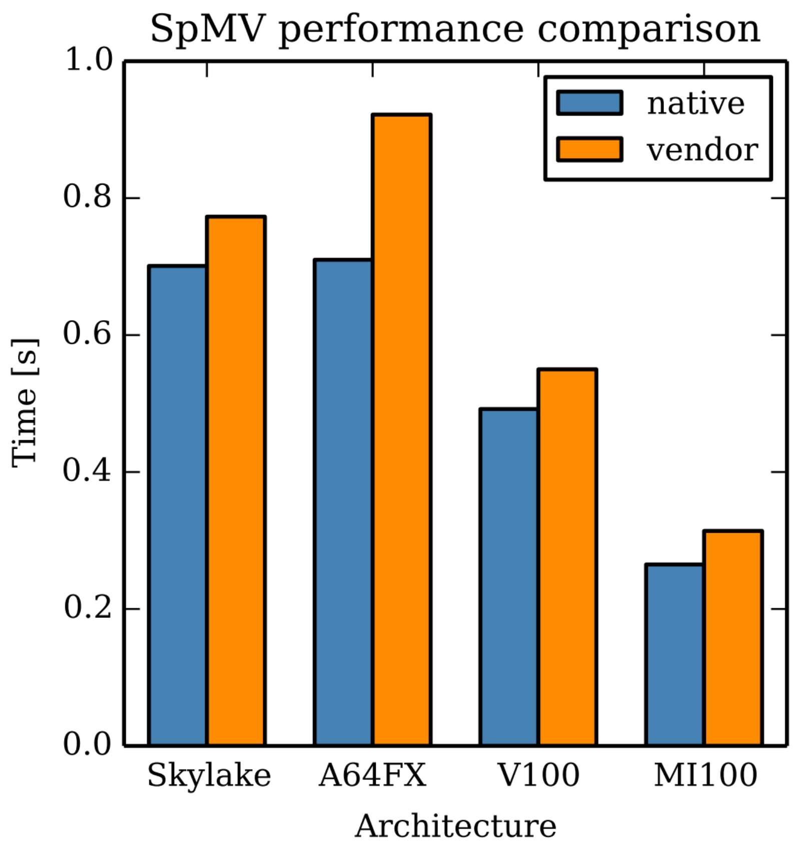
Accelerate <x,A*y> with Kokkos

```
double result;
parallel_reduce(
  "yAx", TeamPolicy(N, AUTO),
  KOKKOS_LAMBDA(auto const &team_handle,
                double &partial_result) {
    int const i = team_handle.league_rank();
    double Ax_i;
    parallel_reduce(
      TeamThreadRange(team_handle, M),
      [&](int const i, double &update) {
        update += A(i, j) * x(j);
      }, Ax_i);
    if (team_handle.team_rank() == 0)
      partial_result += y(i) * Ax_i;
  }, result);
```



Oak Ridge National Laboratory

Sparse Matrix-Vector product (SpMV)



The Kokkos EcoSystem: Comprehensive Performance Portability for High Performance Computing
DOI: 10.1109/MCSE.2021.3098509

Kokkos Kernels native implementation with 3-level hierarchical parallelism competes with vendor optimized libraries

```
parallel_for(                                                            1st level
  "SpMV",
  TeamPolicy((nrows + rows_per_team - 1) / rows_per_team, team_size, 8),
  KOKKOS_LAMBDA(auto const& team) {
   int const first_row = team.league_rank() * rows_per_team;
   int const last_row  = first_row + rows_per_team < nrows
                       ? first_row + rows_per_team : nrows;
   parallel_for(                                                   2nd level
     TeamThreadRange(team, first_row, last_row), [&](int const row) {
      int const row_start  = A.row_ptr(row);
      int const row_length = A.row_ptr(row + 1) - row_start;
      double y_row;
      parallel_reduce(                                       3rd level
        ThreadVectorRange(team, row_length),
        [=](int const i, double& sum) {
          sum +=
            A.values(i + row_start) * x(A.col_idx(i + row_start));
        },
        y_row);
      y(row) = y_row;
     });
  });
```

Wrap up

- We don't want memory management strategy to be dictated
- We want to be able to check accessibility of data
- We want nested algorithms

Thank you!

Damien L-G <lebrungrandt@ornl.gov>