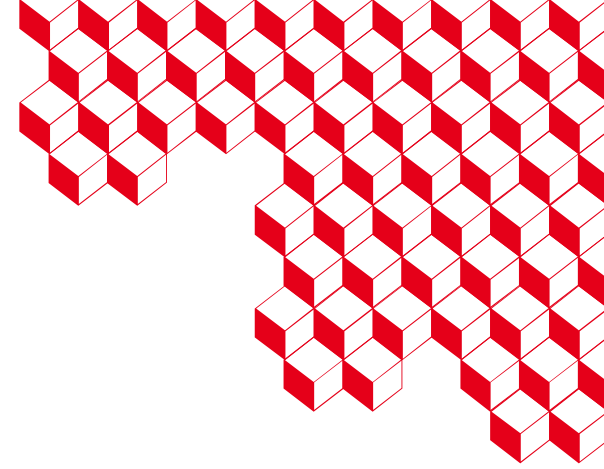




An introduction to the Kokkos ecosystem

Journées prospective calcul de l'IRFU
September 23rd 2026 - Thomas Padioleau & the CExA team



Why Kokkos?

- Challenges in HPC:
 - Exascale leads to hardware **diversity** (CPU, GPU, etc.)
 - Code **duplication** for multiple platforms
- Kokkos offers:
 - Performance portability
 - Modern C++ API
 - Single-source development for multiple backends

Rank	Computer	Vendor	Architecture
1	LineShine	Shenzhen Cloud Computing Center	CPU LX2 304C
2	El capitan	HPE	AMD MI300A
3	Frontier	HPE	AMD MI250X
4	Aurora	Intel	Intel GPU Max
5	JUPITER Booster	EVIDEN	Nvidia GH200

TOP500 June 2026

What is Kokkos?

- A modern C++ library for
 - **Shared memory & data** parallelism
 - Template **meta-programming** and **function objects**
 - Kokkos v5 relies on **C++20**

- Targets major architectures:

- **GPUs**

AMD
ROCm

NVIDIA
CUDA

SYCL

- Multi-core **CPUs**

OpenMP

- Explores cutting-edge architectures

- NextSilicon dataflow accelerators



- Primary contributors:

- Sandia National Laboratories
 - Oak Ridge National Laboratory
 - CEA (CExA)

- Now part of the Linux HPSF

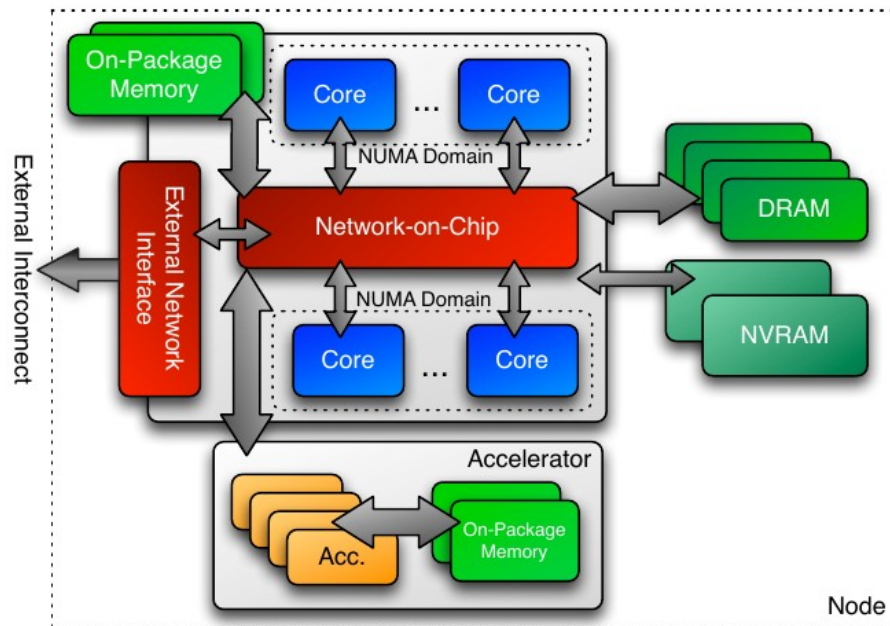


- Large community



Kokkos programming model

- Key idea: **write once, run anywhere**
 - Compilation options to adapt to the hardware
- Abstractions:
 - **Execution spaces**
 - Where instructions run
 - **Memory spaces**
 - Where data lives
 - **Execution policy**
 - How work is scheduled
 - **Multi-dimensional array (View)**
 - Multi-dimensional representation of the data



Parallel patterns

- Loop
 - Kokkos::parallel_for
- Reduction
 - Kokkos::parallel_reduce
- Cumulative reduction
 - Kokkos::parallel_scan
- Dispatch is by default asynchronous
 - Kokkos::fence() to wait for the result

```
parallel_for("for", n, KOKKOS_LAMBDA(int i) {  
    // [...]  
});  
  
parallel_reduce("reduce", n, KOKKOS_LAMBDA(int i, int& local_sum) {  
    local_sum += /* [...] */;  
}, sum_result);  
  
parallel_scan("scan", n, KOKKOS_LAMBDA(int i, int& local_sum, bool  
is_final) {  
    local_sum += /* [...] */;  
    if(is_final) { /* [...] */ }  
});
```

Execution policies: basic parallelism

- Kokkos::RangePolicy
 - 1D **chunked** iteration space
- Kokkos::MDRangePolicy
 - 2-8D **tiled** iteration space
 - Custom iteration order
 - Better cache/coalescing

```
parallel_for(RangePolicy(1, n, chunk_size), KOKKOS_LAMBDA(int i) {  
  
    /* [...] */  
  
});
```

Execution policies: hierarchical parallelism

- Basic parallel patterns can not be nested
- Kokkos::TeamPolicy
 - 2-level nesting is supported by teams of threads
 - Matches groups / threads support in GPU
 - But also available on CPU
 - **Scratch memory** allocation available
- Kokkos::TeamThreadRange
 - Kokkos::ThreadVectorRange
- Kokkos::TeamVectorRange

```
parallel_for(TeamPolicy(nb_teams, team_size), KOKKOS_LAMBDA(const team_handle& team) { /* [...] */  
    parallel_for(TeamThreadRange(team, n, chunk_size), KOKKOS_LAMBDA(int i) { /* [...] */  
        });  
    // [...]  
});
```



Container: Kokkos::View

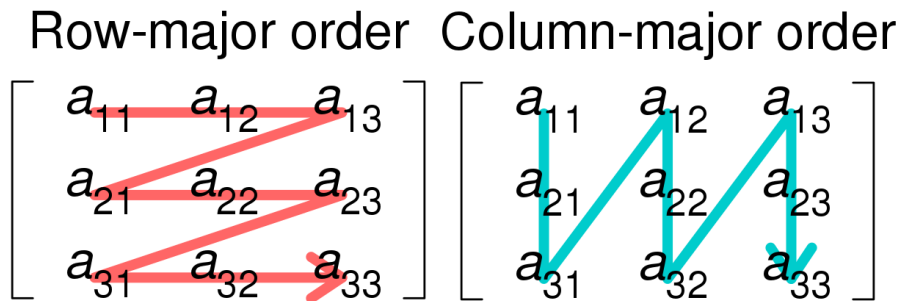
```
template <class DataType [, class LayoutType] [, class MemorySpace] [, class MemoryTraits]>  
class View;
```

- A multi-dimensional array with compile-time parameters
 - Choice of the type of object to store (DataType)
 - Choice of memory layout (LayoutType)
 - Choice of memory space (MemorySpace)
 - Extra information (MemoryTraits)
- Copy semantics
 - Shallow copy, `a = b`, behaves like a `std::shared_ptr`
 - Deep copy, `Kokkos::deep_copy(a, b)`;
 - Mirroring between Host and Device (no-op deep copy if `Device == Host`)
- Compatible with C++23 `std::mdspan`

Memory layout

- `Kokkos::View<double****, Layout> array;`
 - `Kokkos::LayoutRight`
 - `array(i0, i1, i2, i3)`
 - `Kokkos::LayoutLeft`
 - `array(i0, i1, i2, i3)`
 - `Kokkos::LayoutStride`
 - custom ordering
- Important for performance tuning
 - CPU: cache access
 - GPU: coalescent access

$$l = \sum_{\alpha=0}^{n-1} s_{\alpha} i_{\alpha}$$



Mirroring View

- A **mirror view** represents a view in a different memory space
- Similar to its original view (shape, layout, etc.)
- Used to create a mirror of a device view on the host
- *create_mirror*
 - Creates a mirror that always reallocates
- *create_mirror_view*
 - Creates a mirror without allocating memory if the mirror view is allocated in the same memory space as the original view
 - In that case, the mirror view is a shallow copy of the original view (i.e. it targets the same data)

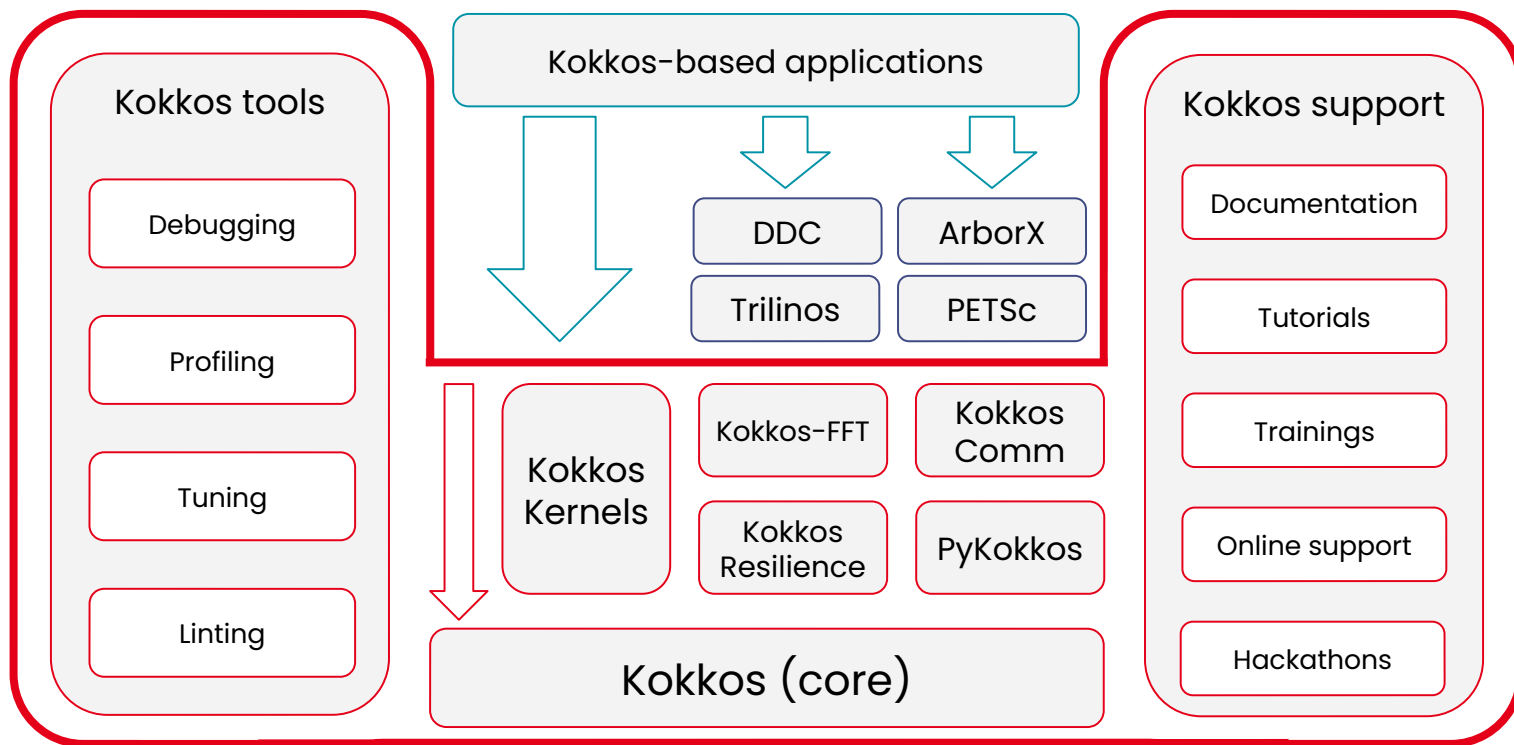
Subview

- A subview is a 'slice' of a View that behaves as a View
 - Same syntax as a View - access data using (multi-)index entries
 - The 'slice' and original View point to the same data - no extra memory allocation or copying
- Can be constructed on host or within a kernel (no allocation of memory occurs)
- Similar capability as provided by Matlab, Fortran, Python, etc. using 'colon' notation

Kokkos core is also...

- Portable atomics
- SIMD classes
- Other containers:
 - OffsetView
 - DualView
 - ScatterView
 - ...
- Algorithms:
 - std-like algorithms
 - Sorting
 - Random number generation
 - Custom reducers
 - ...
- Device compatible standard features:
 - Kokkos::printf
 - Kokkos::numbers
 - Kokkos::pair
 - Kokkos::tuple
 - ...

Kokkos ecosystem and tooling



Introduction to Kokkos on CPU and GPU

Kokkos-FFT



- What is it?
 - Fast Fourier Transform interface integrated into the Kokkos ecosystem
 - Enables portable FFT computations on heterogeneous systems
- Why it matters?
 - FFT libraries are usually hardware-specific
 - Managing CPU/GPU differences is complex
- Key Features
 - Latest release v2
 - Supports 1D, 2D, and 3D FFTs
 - Works directly with Kokkos Views
 - Backend abstraction (e.g., FFTW, cuFFT, hipFFT)

KREPE

- What is it KREPE?
 - KREPE = Kernel Replay, Execution for Performance Evaluation
 - A Kokkos kernel replayer that captures the execution context of a kernel and replays it in a standalone program
 - Developed by the **CExA** project in collaboration with **Dyablo/IRFU** to enable controlled and reproducible performance evaluation of Kokkos kernels
- Why KREPE?
 - Isolate kernels: evaluate an individual kernel without running the complete application
 - Reproduce executions: replay the same kernel with the original input data and execution policy
 - Benchmark & tune: change the execution policy or hardware configuration and compare performance
 - Simplify performance analysis: capture the kernel's relevant data once, then experiment independently
- Key features
 - Capture & replay: dump a Kokkos kernel's input/output memory state and replay it later
 - Execution-policy preservation: records policies such as RangePolicy, MDRangePolicy and TeamPolicy
 - HDF5-based dumps: stores allocation data, metadata and execution-policy information in portable dump files
 - Kokkos integration: designed to work directly with Kokkos and built through CMake



Kokkos Comm

- What is Kokkos Comm?
 - A lightweight C++ library for distributed communication on Kokkos
 - Designed for performance portability across heterogeneous systems (CPU, GPU, multi-node)
- Why Kokkos Comm?
 - Kokkos handles intra-node parallelism, but inter-node communication is still complex
 - Traditional MPI usage with Kokkos leads to:
 - Code duplication
 - Manual data packing/unpacking
 - Difficult GPU-aware communication
- Key Features
 - Latest release v0
 - Performance-portable communication API
 - Asynchronous operations (non-blocking send/recv, collectives)
 - Automatic handling of:
 - GPU-aware communication
 - Non-contiguous data (Kokkos Views)
 - Minimal, composable, modern C++20 interface
 - Built on top of existing communication libraries:
 - MPI, NCCL (others under exploration)

Kokkos Kernels

- What is it?
 - Library of optimized parallel kernels built on Kokkos
 - Provides standard HPC building blocks (BLAS, sparse, graph)
- Why it matters?
 - Avoid reimplementing performance-critical kernels
 - Tuned for CPUs, GPUs, and manycore systems
 - Ensures portability without sacrificing performance
- Key Features
 - Latest release v5
 - Dense linear algebra (BLAS-level operations)
 - Sparse matrix operations (SpMV, SpGEMM, triangular solves)
 - Graph kernels and utilities

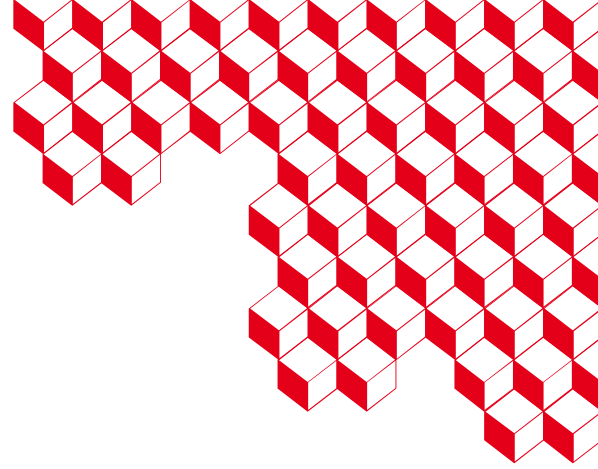


Further reading and references

- Kokkos 3: Programming Model Extensions for the Exascale Era
- The Kokkos Ecosystem: Comprehensive Performance Portability for High Performance Computing
- Kokkos [GitHub Repository](#)
- Kokkos [Documentation](#)
- Kokkos [cheatsheet](#)
- Kokkos [Tutorials](#) (YouTube), [Tutorial](#) (CExA version)

Conclusion

- Kokkos enables performance portability
- Abstracts hardware-specific details
- Easy to adopt in modern C++ codebases
- Widely supported and actively developed
- **CEA invests in the development of the Kokkos ecosystem**



Thank you for your attention

Questions ?

Introduction to Kokkos on CPU and GPU