Kokkos, a sustainable performance portability layer

co-developed by CExA, a project for Computing at Exascale on Accelerators at CEA


Adrien Taberner
Developer


Dr. Ansar Calloo
Group leader, Senior Developer


Dr. Cédric Chevalier
Group leader, Senior developer


Dr. Hariprasad Kannan
Developer


Dr. Mathieu Lobet
Group leader, Senior developer


Dr. Paul Zehner
Developer


Dr. Rémi Baron
Senior Developer

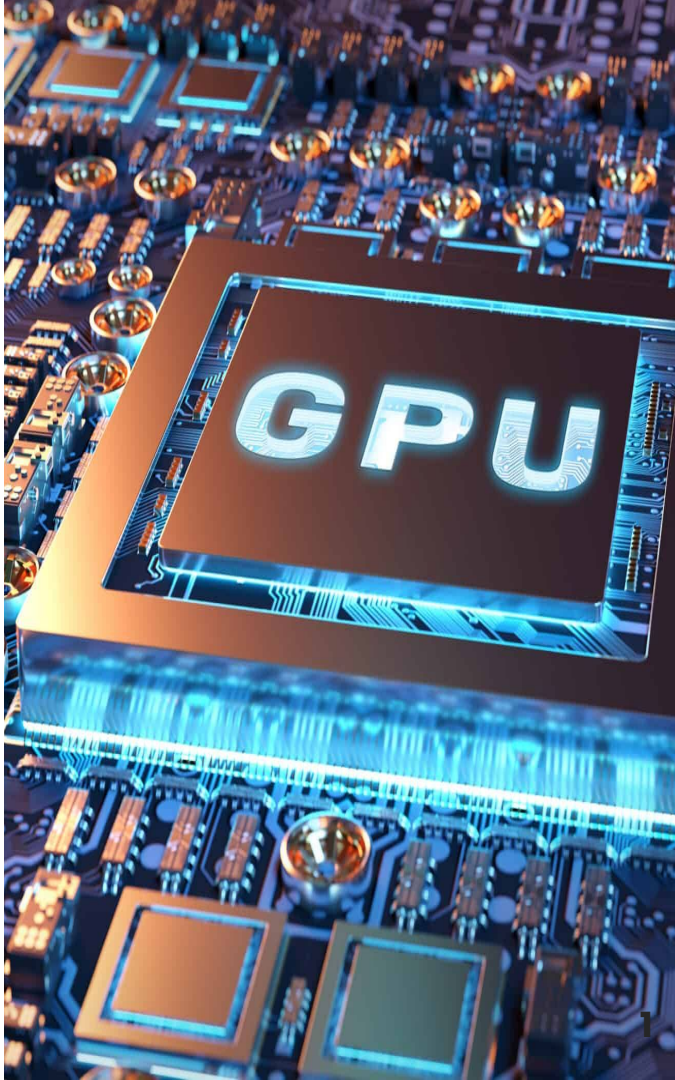

Dr. Thomas Padioleau
Senior developer, architect


Dr. Yuuichi Asahi
Senior developer


Paul Gannay
Developer

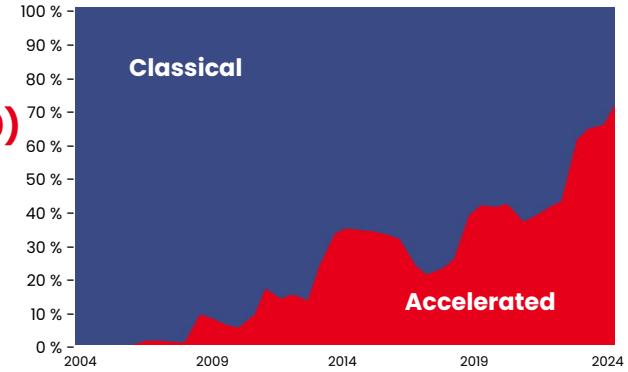

Trévis Morvany
Developer

ORAP Forum
May 19th 2025
Julien Bigot, the CExA, Kokkos & HPSF teams

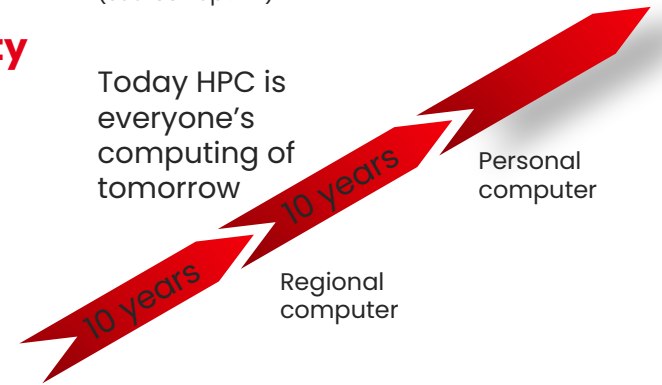


Context 1+ year ago

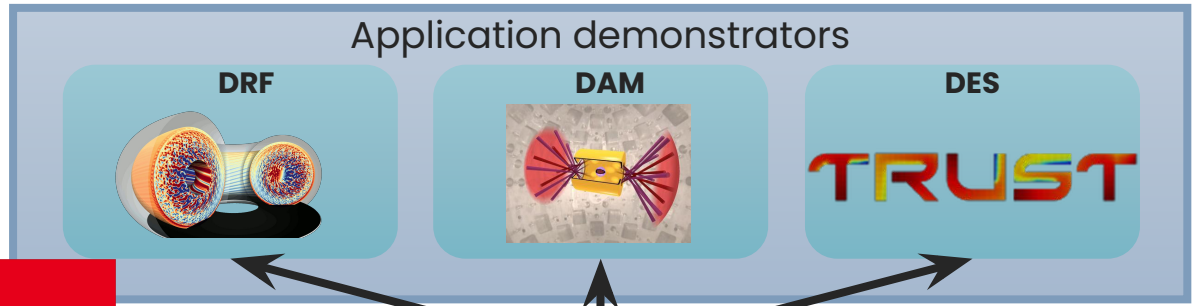
- HPC is a tool in a wide range of domains at CEA
- We just entered the **Exascale** era, that means **GPU (>82% Top500)**
  - European pre-Exascale systems: Mix of AMD & Nvidia
  - US Exascale systems: AMD & Intel
  - First European Exascale machines are coming
    - *Jupiter* at Jülich (Germany) => Nvidia & Rhea
    - *Alice Recoque* at **CEA/TGCC** (call still open)
  - Need to re-develop applications with **Performance portability**
- GPU programming models: **software catalysts**
  - France and Europe: great research but no production tool
- A **need** for a long-term sustainable solution
  - **Adapted** to our hardware and software specificities
  - Where we can have **trust** in the roadmap

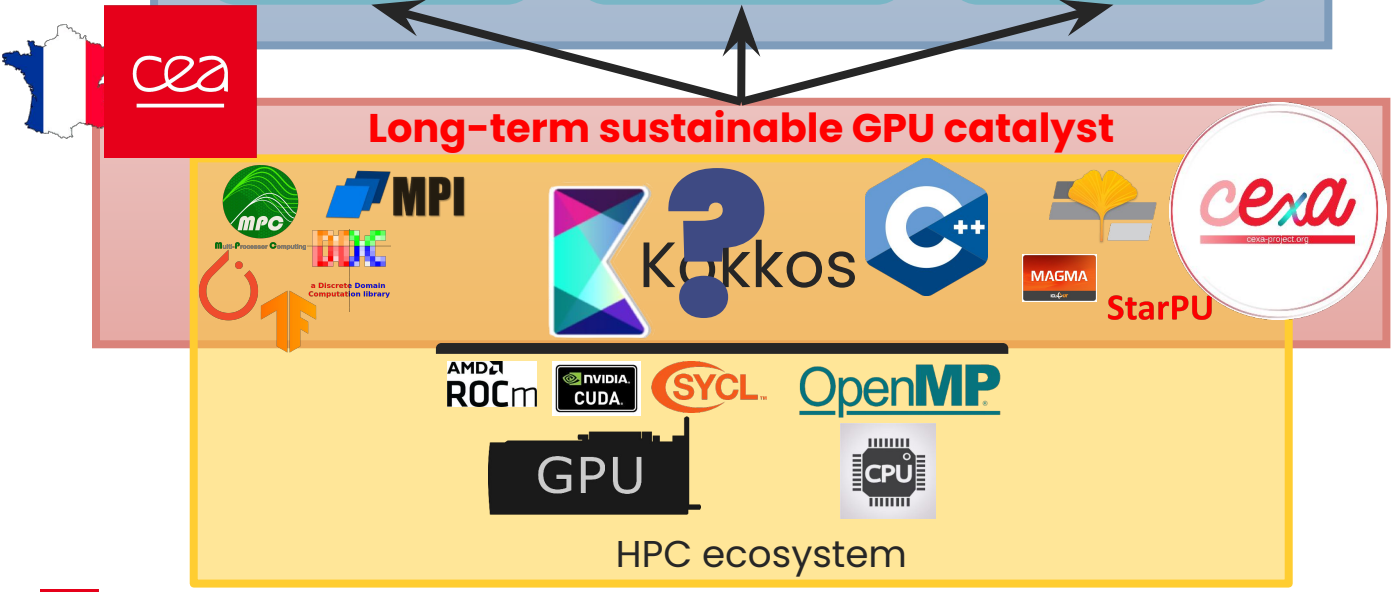


Computing power of the 500 top supercomputers from june 2004 to june 2024 (source Top500)



CExA project: goals





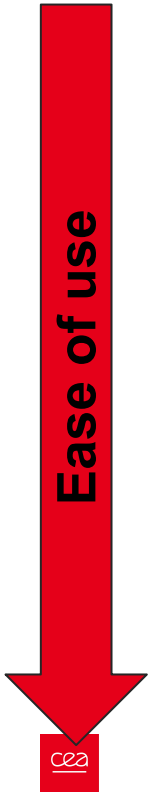
Disseminate and offer training

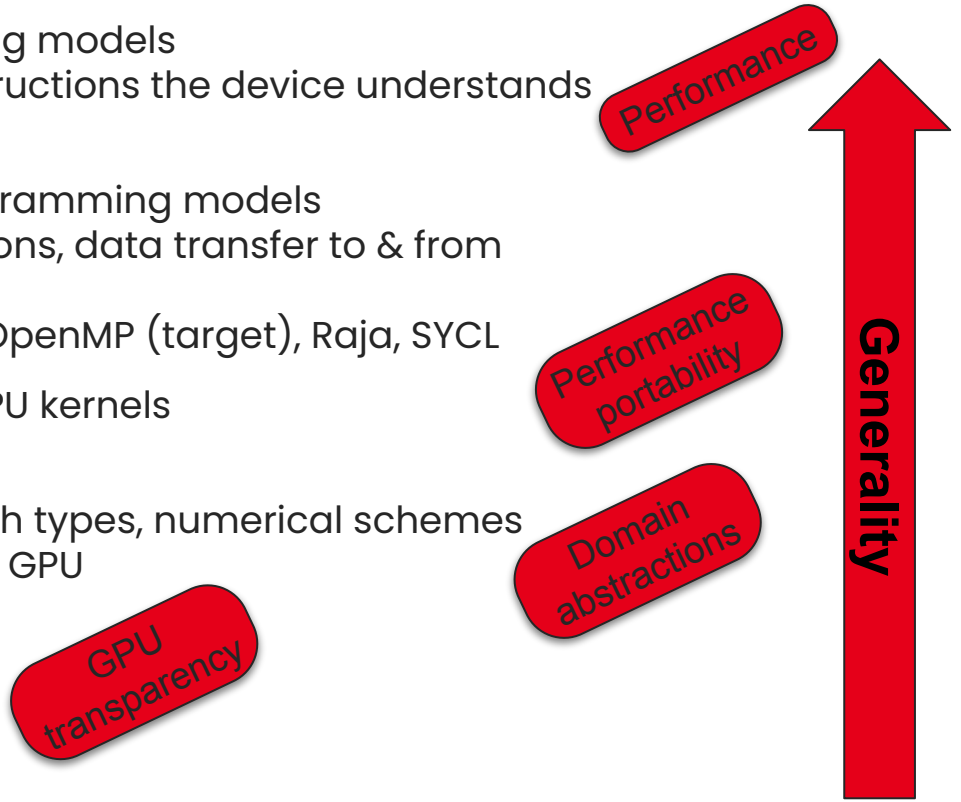
Adapt application demonstrators

Provide a long-term sustainable software catalyst for GPU computing

How to write code for GPU?

- Low-level, assembly-style programming models
  - Nearly manipulate the actual instructions the device understands
  - E.g. HSA, Level Zero, PTX, Spir-V , …
- General-purpose, imperative GPU programming models
  - Manipulate parallel loops, reductions, data transfer to & from device
  - E.g. Cuda, HIP, Kokkos, OpenACC, OpenMP (target), Raja, SYCL
- Combination & assembly of existing GPU kernels
  - Pytorch, StarPU, etc…
- Application framework for specific mesh types, numerical schemes
  - Use domain-specific concepts on GPU
- Pre-written GPU libraries
  - just call them from CPU
  - Neural Networks, Linear Algebra, …





How to write code for GPU?

- Low-level, assembly-style programming models
  - Nearly manipulate the actual instructions the device understands
  - E.g. HSA, Level Zero, PTX, Spir-V , …
- General-purpose, imperative GPU programming models
  - Manipulate parallel loops, reductions, data transfer to & from device
  - E.g. Cuda, HIP, Kokkos, OpenACC, OpenMP (target), Raja, SYCL
- Combination & assembly of existing GPU kernels
  - Pytorch, StarPU, etc…
- Application framework for specific mesh types, numerical schemes
  - Use domain-specific concepts on GPU
- Pre-written GPU libraries
  - just call them from CPU
  - Neural Networks, Linear Algebra, …

Ease of use

Performance

Performance portability

Domain abstractions

GPU transparency

Generality

How to write code for GPU?

- Low-level, assembly-style programming models
  - Nearly manipulate the actual instructions the device understands
  - E.g. HSA, Level Zero, PTX, Spir-V , …
- General-purpose, imperative GPU programming models
  - Manipulate parallel loops, reductions, data transfer to & from device
  - E.g. **Cuda, HIP, Kokkos, OpenACC, OpenMP (target), Raja, SYCL**
- Combination & assembly of existing GPU kernels
  - Pytorch, StarPU, etc…
- Application framework for specific mesh types, numerical schemes
  - Use domain-specific concepts on GPU
- Pre-written GPU libraries
  - just call them from CPU
  - Neural Networks, Linear Algebra, …

Ease of use

Generality

Performance

Performance portability

Domain abstractions

GPU transparency

Available solutions

- Cuda
- HIP
- Kokkos
- OpenACC
- OpenMP (target)
- Raja
- SYCL
  - OneAPI/DPC++
  - AdaptiveC++/OpenSYCL/hipSYCL

Available solutions

- Cuda
- HIP
- Kokkos
- OpenACC
- OpenMP (target)
- **Raja**
- SYCL
  - OneAPI/DPC++
  - **AdaptiveC++/OpenSYCL/hipSYCL**

- **Production grade, with public support**

Available solutions

- **Cuda**
- **HIP**
- Kokkos
- **OpenACC**
- OpenMP (target)
- Raja
- SYCL
  - **OneAPI/DPC++**
  - AdaptiveC++/OpenSYCL/hipSYCL

- Production grade, with public support

- **Vendor neutral**

Available solutions

- Cuda
- HIP
- **Kokkos**
- OpenACC
- **OpenMP (target)**
- Raja
- SYCL
  - OneAPI/DPC++
  - AdaptiveC++/OpenSYCL/hipSYCL

- Production grade, with public support
- Vendor neutral

OpenMP & Kokkos : the simplest GPU loop

```
for (int j = 0 ; j < Nj ; ++j) {
      // [...]
}
```

Sequential

```
#pragma omp teams distribute parallel for
for (int j = 0 ; j < Nj ; ++j) {
      // [...]
}
```

OpenMP Target

```
parallel_for(Nj, KOKKOS_LAMBDA(int j) {
      // [...]
});
```

Kokkos

Execute in parallel, on a separate GPU thread each,

the same workload *[. . .]*

identified by a unique identifier **j**

Nj times between 0 and Nj-1

OpenMP & Kokkos : memory transfer

```
double* x = malloc(Ni*sizeof(double));
double* y = malloc(Nj*sizeof(double));
double* A = omp_target_alloc(
      Ni*Nj*sizeof(double),
      omp_get_initial_device());

#pragma omp target data \
      map(to: x[0:Ni]) \
      map(from: y[0:Nj])
{
#pragma omp teams distribute parallel for
for (int j = 0 ; j < Nj ; ++j) {
      for (int i = 0 ; i < Ni ; ++i) {
            y[j] += x[i] * A[j*Ni+i];
      }
}
```

OpenMP Target

```
View<double*, Kokkos::HostSpace> x(Ni);
View<double*, Kokkos::HostSpace> y(Nj);
View<double*> A(Nj, Ni);

{
auto dx = create_mirror_view_and_copy(dev, x);
auto dy = create_mirror_view(dev, y);
parallel_for(Nj, KOKKOS_LAMBDA(int j) {
      for (int i = 0 ; i < Ni ; ++i) {
            dy(j) += dx(i) * A(j,i);
      }
});
deep_copy(y, dy);
}
```

Kokkos

Copy x to GPU from device before kernel
and y from GPU to device after kernel
Keep A on the device

Compilation

OpenMP Target

- Use an OpenMP compiler
  - Compatible with the target construct
  - Compatible with the hardware you target
- Each vendor provides its own OpenMP compiler
  - Usually based on LLVM infra
- Default Clang/LLVM & GCC also try to support this
  - For some hardware
  - With variable performance

Kokkos

- A C++ template library
  - No direct code generation, rely on vendors C++-like languages
- Multiple “backends”, selection at compile time
  - Each vendor’s preferred toolchain
  - OpenMP, Cuda, OneAPI, HIP, …
- Maximum 3 backends enabled at once
  - Serial backend
  - 1 Host parallel backend (openmp)
  - 1 Device parallel backend (cuda, HIP, Sycl)

Available solutions

- Cuda
- HIP
- **Kokkos**
- OpenACC
- **OpenMP (target)**
- Raja
- SYCL
  - OneAPI/DPC++
  - AdaptiveC++/OpenSYCL/hipSYCL

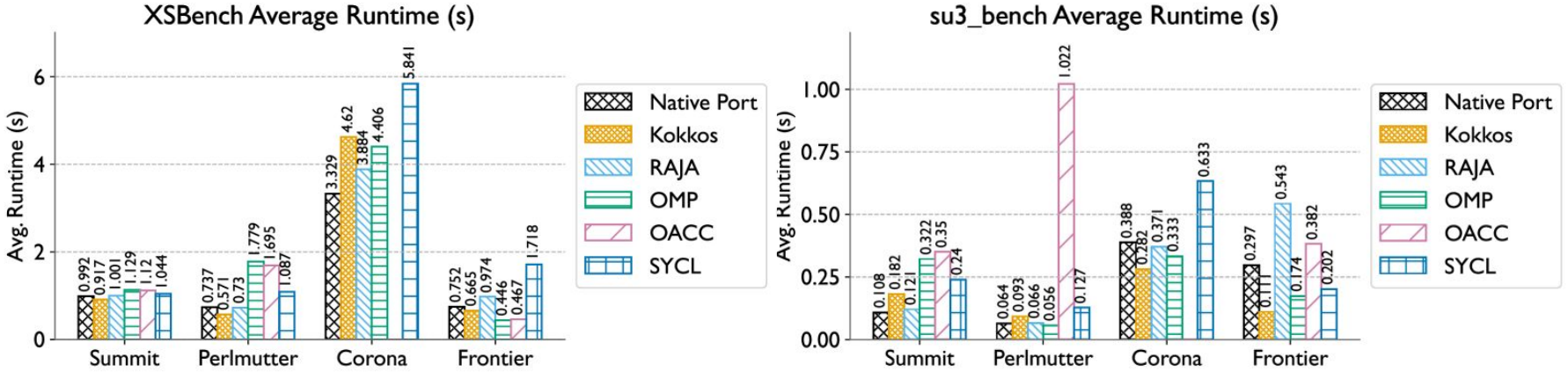
- Production grade, with public support
- Vendor neutral

Available solutions

- Cuda
- HIP
- **Kokkos**
- OpenACC
- **OpenMP (target)**
- Raja
- SYCL
  - OneAPI/DPC++
  - AdaptiveC++/OpenSYCL/hipSYCL

- Production grade, with public support
- Vendor neutral
- Annotations
  - Works best with imperative languages: C, Fortran, …
  - Compiler integration: potential for additional optimizations
  - Seq. first, requires to re-design applications for GPU
- Library
  - Suited to language with deep encapsulation: C++, …
  - On top of vendor toolchains: easier to port to new hardware
  - GPU first, requires to re-write applications for GPU

And what about performance?



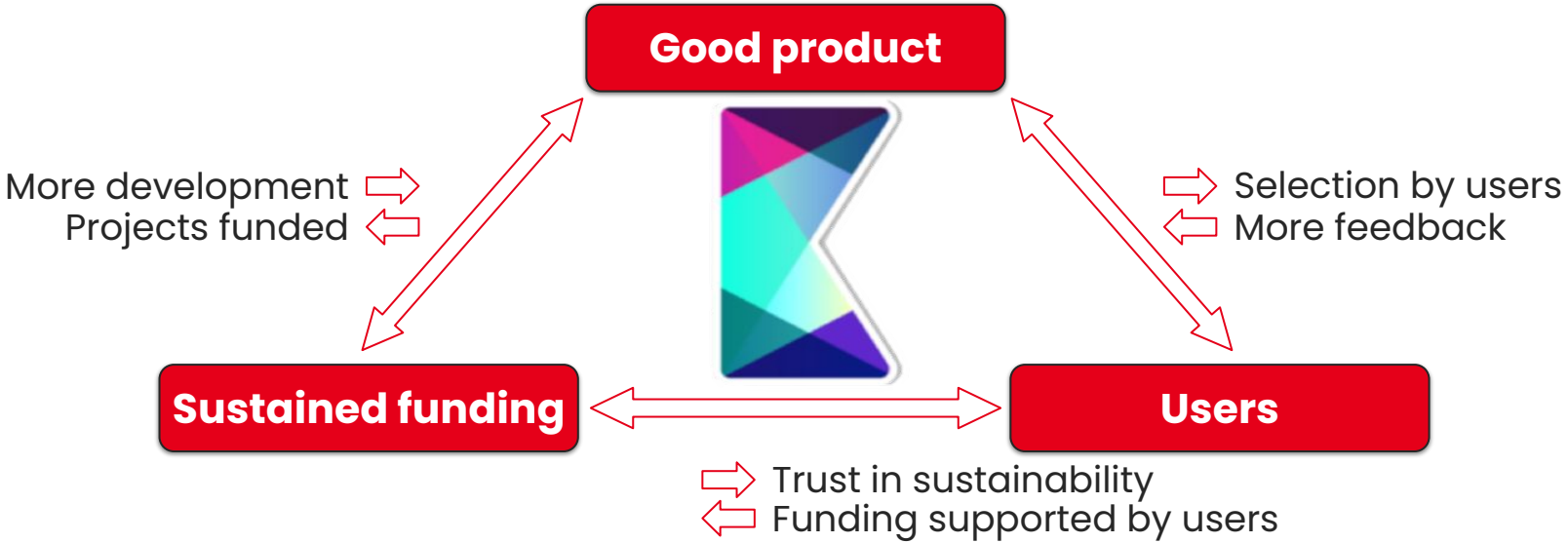
Average runtime of the XSBench (left) and su3_bench (right) proxy apps across all platforms and programming models.
Lower is better.

An Evaluative Comparison of Performance Portability across GPU Programming Models

Joshua H. Davis[2], Pranav Sivaraman[2], Isaac Minn[2], Konstantinos Parasyris[1], Harshitha Menon[1], Giorgis Georgakoudis[1], Abhinav Bhatele[2]

[2]Department of Computer Science, University of Maryland
[1]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

Kokkos at the center of a virtuous cycle

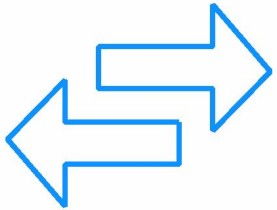


There is strength in numbers:
collaboration on core products is good for everyone

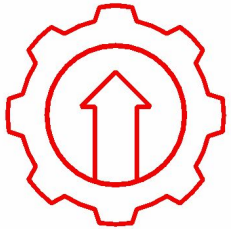
Kokkos, a HPSF project

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Performance



Portability



Productivity

1. A neutral hub for open source, high performance software
2. Supports projects that advance portable software for diverse hardware by:
   - Increasing adoption
   - Aiding community growth
   - Enabling development efforts
3. Lower barriers to productive use of today's and future high performance computing systems

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Fund & vote

Members

Premier

Hewlett Packard Enterprise
Sandia National Laboratories
aws
Lawrence Livermore National Laboratory

General

AMD
Argonne NATIONAL LABORATORY
arm
cea
intel
kitware Delivering Innovation
NVIDIA
Los Alamos NATIONAL LABORATORY EST.1943
OAK RIDGE National Laboratory
RIKEN
R-CCS

Associate

BERKELEY LAB
C-DAC
JÜLICH Forschungszentrum
Tennessee TECH
UCL
UNIVERSITY OF MARYLAND
UNIVERSITY OF OREGON
UTokyo
Universität der Bundeswehr München

aws Hewlett Packard Enterprise | Sandia National Laboratories | Lawrence Livermore National Laboratory | cea | kokkos

Governing board

Participate & vote

Technical Advisory Council

Join & vote

WGs & committees

- Outreach
- Diversity
- CI & Testing
- Events
- Tools
- ...

Projects

Spack
Charliecloud
AMReX
HPCToolkit
APPTAINER
E4S
TRILINOS
kokkos
OpenCHAMI
VISKORES
HPX
WarpX
ENVIRONMENT MODULES
CHAPEL

The TAC has established a project lifecycle as a path to sustainability

Emerging

- Committed to open governance
- Working towards best practices
- Important projects for the HPC ecosystem

Established

- Wide usage by at least 3 orgs of sufficient size and scope
- Steady commits from at least one organization
- Robust development practices

Core

- Used commonly in production environments
- Steady commits from *more* than one organization
- Large, well-established project communities
- Sustainable cycle of development and maintenance

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Neutrality through HPSF

1. Sustaining OSS projects requires a community
2. Building a community requires trust
    - Projects will continue to be available
    - Projects are usable by anyone
    - No one organization can control the direction of the project
    - Projects are open to new contributors and new ideas
3. Trust gets us users; some users become contributors
4. Neutral, open governance ensures that we can build the broadest possible communities

The CExA project

“adopt and adapt” strategy based on Kokkos

Kokkos : **a strong technical basis**

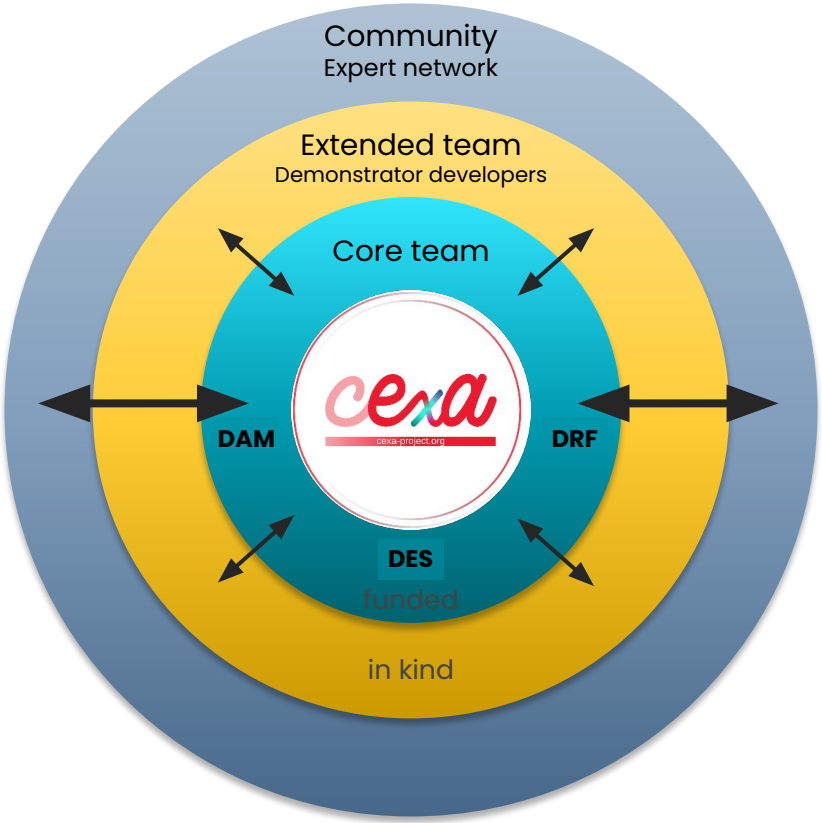
- A software architecture ready for the future
- Mature, free, libre, and open-source
- An **independent foundation** to own the product
  - HPSF under the Linux Foundation

- A standardisation effort in ISO C++
  - A **stepping stone** one step ahead toward **parallel C++**

Some **adaptations required**

- For European **hardware**
  - There is no real hardware sovereignty without software sovereignty
- For **applications** from CEA, France and Europe
  - Take our specificities into account

CExA project in practice



- **Core team**
  - Management, implementation and dissemination
  - 12 researchers from all over CEA
  - 6 dedicated recrutements
    - 1 as a permanent researcher !
  - Funding for 2 or 3 hires expected every year
- **Extended team**
  - Demonstrator developers
    - Not funded
    - Find their own interest in the participation
  - 2-3 new demonstrators every year
- **Community**
  - Federation of an expert network
  - Co-design of C**ExA**:
    - Identification of needs
    - Usage of C**ExA** in applications
  - Priority target for dissemination
  - Sustainability of the work

Kokkos development today

Primary teams

Contributions & support

- Number of commits by institution
- In the last 6 month

Kokkos parallel patterns

```
parallel_for(Nj, KOKKOS_LAMBDA(int j) {
      // [...]
});
```

Kokkos parallel patterns

```
parallel_for(Nj, KOKKOS_LAMBDA(int j) {
      // [...]
});
```

```
parallel_reduce(Nj, KOKKOS_LAMBDA(int j, double& accumulator) {
      // [...]
      accumulator += /* [...] */ ;
}, result);
```

```
parallel_scan(Nj, KOKKOS_LAMBDA(int j, double& result, bool isfinal)
{
      // [...]
      accumulator += /* [...] */ ;
      if(is_final) {
            // [...]
      }
}, result);
```

- For
  - independent iterations
- Reduce
  - Accumulate into a single value
- Scan
  - N independent prefix reduction

Kokkos parallel patterns: easy debug

```
parallel_for("loop1", Nj, KOKKOS_LAMBDA(int j) {
      // [...]
});
```

- Naming loops ease debugging & profiling
- Integrated with kokkos-specific tools
- Get a trace with names includes
- Get a name in debug messages
- Omitted in the presentation, but a good practice overall

Kokkos parallel patterns: ExecutionSpace

```
parallel_for<ExecutionSpace>(Nj, KOKKOS_LAMBDA(int j) {
      // [...]
});
```

- ExecutionSpace defines where to run
  - `Cuda`, `HIP`, `SYCL`, `HPX`, `OpenMP`, `OpenMPTarget`, `Threads`, `Serial`
  - 3 exec spaces per execution max: Serial + parallel Host + parallel Device
- Choose where to run at compile time with a `#define`
  - Usually set from CMake
- 2 predefined aliases are often enough
  - DefaultExecutionSpace: parallel Device, or parallel Host, or Serial
    - Most of the time
  - DefaultHostExecutionSpace: parallel Host, or Serial
    - When using host-only code

Kokkos parallel patterns: Policies

```
parallel_for(RangePolicy(DefaultExecutionSpace(), 1, Nj, chunk_size), KOKKOS_LAMBDA(int j) {
      // [...]
});
```

Beyond simple 1D execution

- Give an instance of ExecutionSpace for multi-GPU or multi-Stream support
- RangePolicy for 1D iteration
  - Begin / end iteration boundaries
  - Chunk_size hint for improved performance
- MDRange policy for multi-dimensional iterations
  - Multi-D begin / end iteration boundaries
  - Tiling hint hint for improved performance

Kokkos parallel patterns: hierarchical parallelism

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

- Default loops can not be nested
- 2-level nesting is supported by teams of threads
  - Matches groups / threads support in GPU
  - But also available on CPU
  - Intermediate (scratch) memory allocation available

Kokkos parallel patterns are asynchronous

```
parallel_for(Nj, KOKKOS_LAMBDA(int j) {
      // [...]
});
parallel_for(Nj, KOKKOS_LAMBDA(int j) {
      // [...]
});
fence();
```

- Asynchronous execution
- Result visibility is only assured after a fence
- Or between kernels running on the same execution space

Kokkos views: multi-dimensional arrays

```
View<int**, MemorySpace> my_matrix("matrix", Nx, Ny);
```

- Multi-dimensional arrays
  - Type & dimensionality specified: `int**` => 2D integer array
  - Dynamic sizes are parameters: `Nx, Ny`
  - Static sizes are also possible: `int*[4]` => 2D array, 4 × dynamic
- Behaves like a C++ `shared_ptr`
  - Shared ownership with reference counting (like in python)
- With a name for debugging/profiling
- MemorySpace is part of the type, defaults should be used
  - `CudaSpace, CudaHostPinnedSpace, CudaUVMSpace, HIPSpace, HIPHostPinnedSpace, HIPManagedSpace, SYCLDeviceUSMSpace, SYCLHostUSMSpace, SYCLSharedUSMSpace,` **`HostSpace`**`,` **`SharedSpace`**`, SharedHostPinnedSpace`
  - Check of accessibility between MemorySpace & ExecutionSpace

Kokkos views copies & co.

```
auto dview = subview(oview, pair(start, end), ALL, slice_idx);
```

- Make a new reference to a subset of an existing view
  - Modifying the result modifies the source
  - pair: select a subrange, ALL: keep the dimension, integer: slice the dimension

```
void deep_copy(const ExecSpace &exec_space, const ViewDest &dest, const ViewSrc &src);
```

- Copy data between 2 views
  - Potentially on distinct memory spaces
  - An asynchronous operation

```
auto dview = create_mirror(mspace, a_view); // allocates & copy a new view of same size
auto dview = create_mirror_view_and_copy(mspace, a_view); // allocates & copy if necessary
```

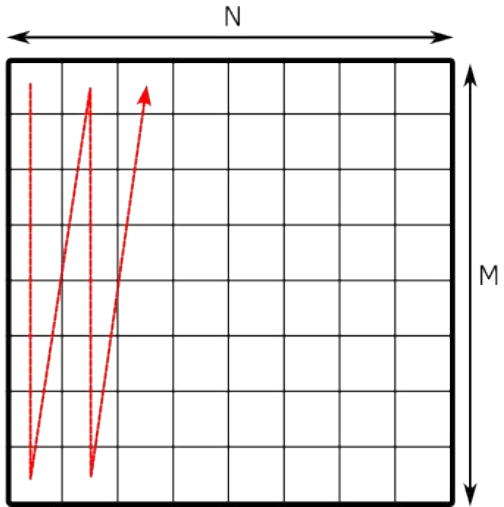
- Allocates & copy to a new memory space

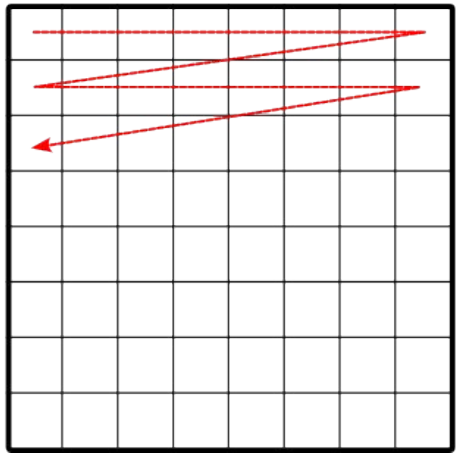
Kokkos views layout

```
View<double**, LayoutLeft> A("A", M, N);
```

- Layout specifies the linearization of multi-D indices into memory
  - `LayoutLeft` (a.k.a Fortran, default on GPU)
  - `LayoutRight` (a.k.a C, default on Host)
  - `LayoutStride` (generic, useful for subviews)

Layout left
Column major in 2D
(Fortran)
Device default layout





Layout right
Raw major in 2D
(C, C++, Python, Java)
Host default layout

What's in Kokkos (core library)?

Multi-dimensional arrays

- Layout auto change for performance

Parallel patterns w. asynchronous support

- Independent interactions, Reductions, Scans

Iteration strategies

- Tiled, Hierarchical, ...

What's in Kokkos (core library)?

Multi-dimensional arrays

- Layout auto change for performance

Other containers

- Key-value maps, ScatterView …

Automatic ref-counted Host/Device memory allocation & management

Host/device memory transfers

Support of "dual" arrays with one version on each side

- Up-to-date tracking & automatic transfers when required

Scratch memory

- Using "team-local" fast memory on the device

Parallel patterns w. asynchronous support

- Independent interactions, Reductions, Scans

Iteration strategies

- Tiled, Hierarchical, …

Algorithms

- Sorting
- Random number generation
- Many of STL parallel algorithms
- …

QoL features: portable printf, etc.

Portable atomic operations

SIMD

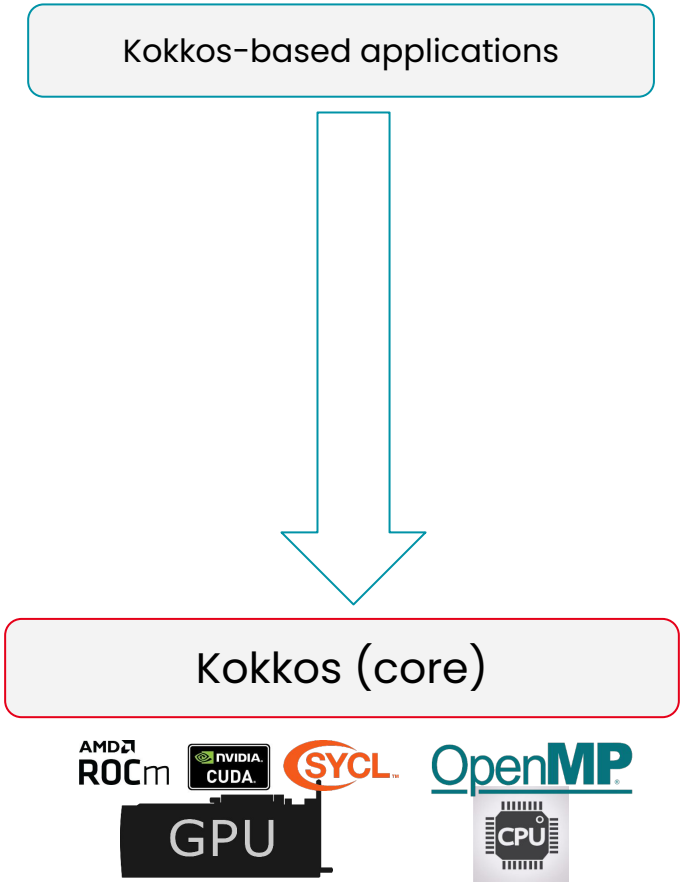
Coarse & fine-grain tasks

And much more…

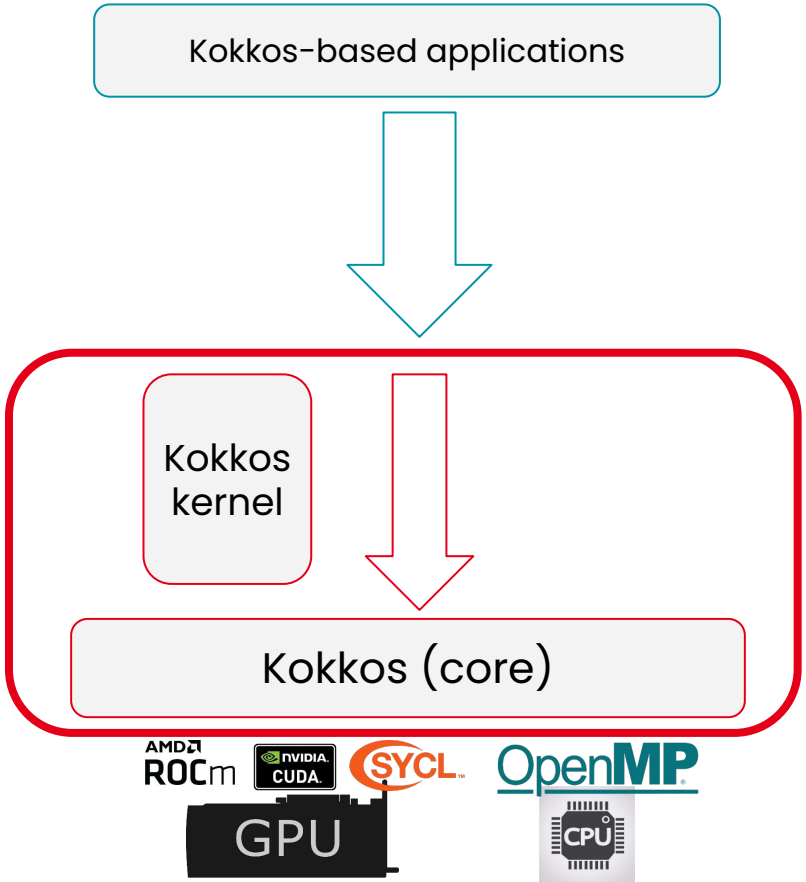
What's new in Kokkos (core library)?

- Multi-GPU support for CUDA, HIP, and SYCL
  - Launch kernels on multiple devices from a single host process
- Many Team-Level Algorithms
  - Extended API with a new overload for team-level support
- Build systems update
  - support linking against build tree & building with hidden visibility
- New Kokkos::View implementation
  - Now mdspan-based
- Improved sorting performance
- Updated SIMD support to align with C++26
- Improved Thread-Safety
  - Kernels can be submitted from multiple threads to the same execution space
- Clarified support for View of Views
  - Introduced a new `SequentialHostInit` view allocation property
- Improved clang-tidy Conformity

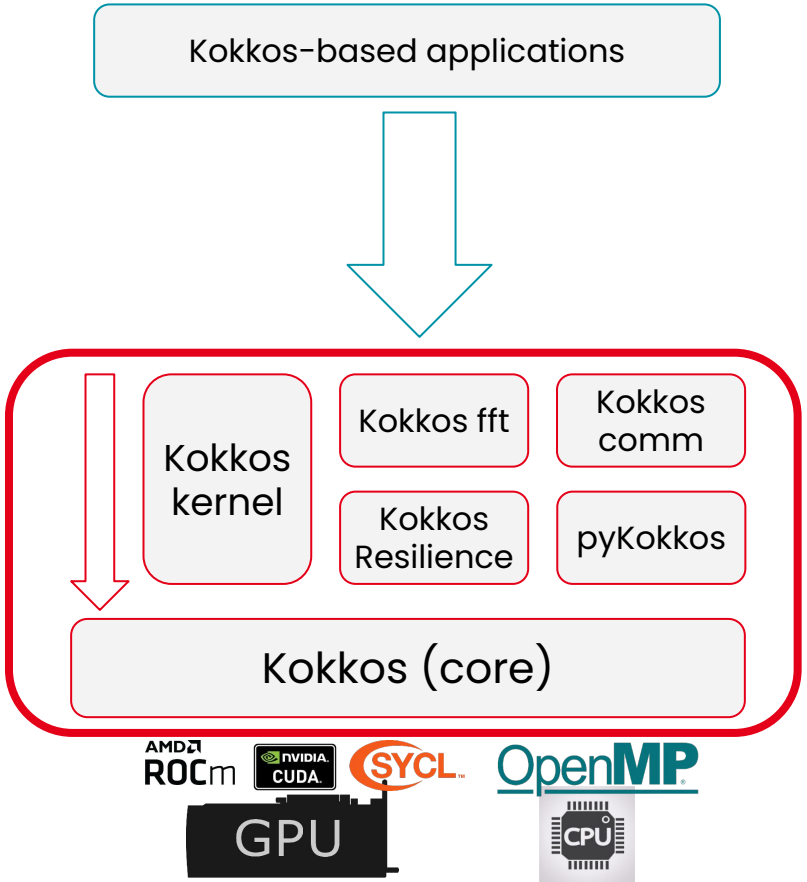
Kokkos Ecosystem



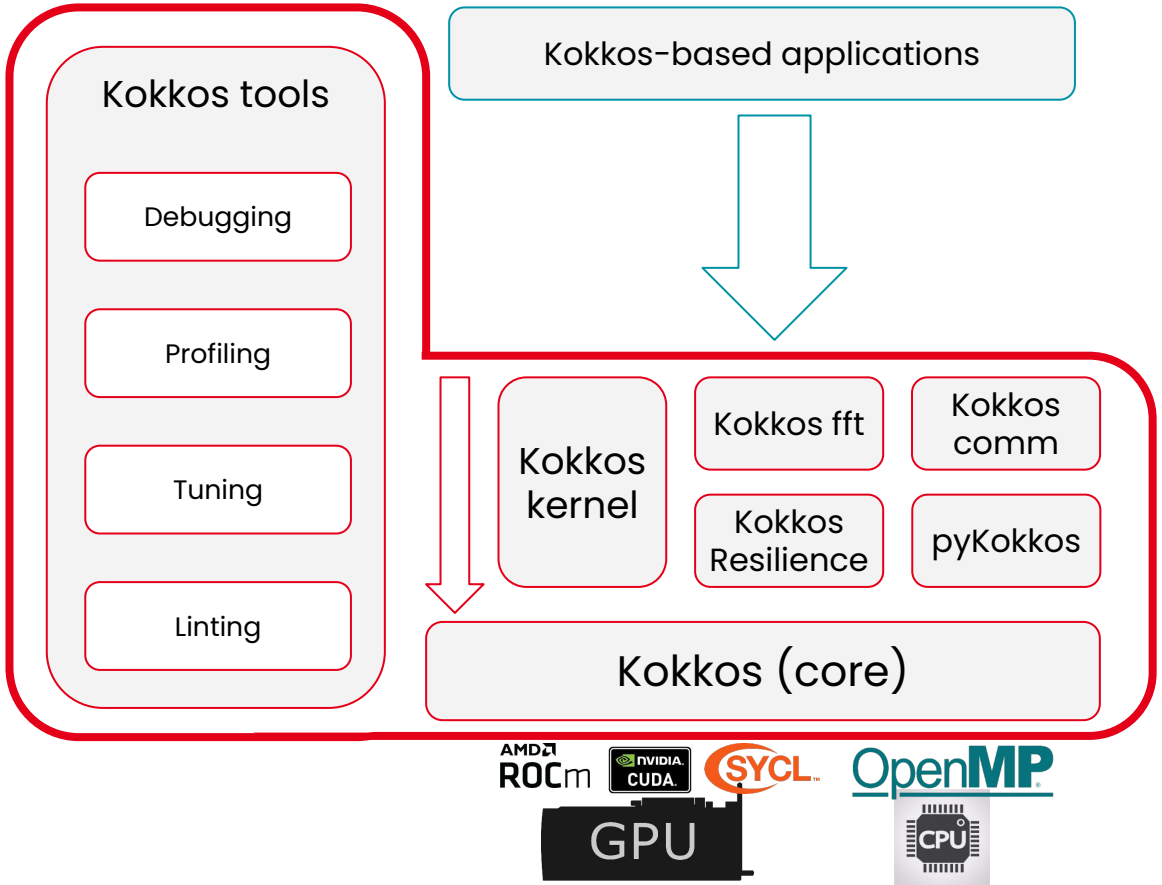
Kokkos Ecosystem



Kokkos Ecosystem



Kokkos Ecosystem



Kokkos Ecosystem

Kokkos tools

- Debugging
- Profiling
- Tuning
- Linting

Kokkos-based applications

Kokkos kernel

Kokkos fft

Kokkos comm

Kokkos Resilience

pyKokkos

Kokkos (core)

Kokkos support

- Documentation
- Tutorials
- Trainings
- Online Support
- Hackatons

AMD ROCm NVIDIA CUDA SYCL OpenMP GPU CPU

Kokkos Ecosystem

Kokkos tools

- Debugging
- Profiling
- Tuning
- Linting

Kokkos-based applications

- DDC
- ArborX
- Trilinos
- PETSc

Kokkos kernel

- Kokkos fft
- Kokkos comm
- Kokkos Resilience
- pyKokkos

Kokkos (core)

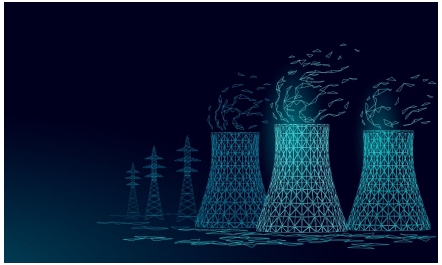
Kokkos support

- Documentation
- Tutorials
- Trainings
- Online Support
- Hackatons

AMD ROCm · NVIDIA CUDA · SYCL · OpenMP

GPU · CPU

To conclude



- Kokkos is a strong vendor-neutral, performance portable Exascale programming model with GPU support
- A worldwide collaborative effort for an application first strategy
- CExA & HPSF ensure it is a sovereign and sustainable approach that can be relied on for the foreseeable future
- A strong dynamic all over the CEA and beyond