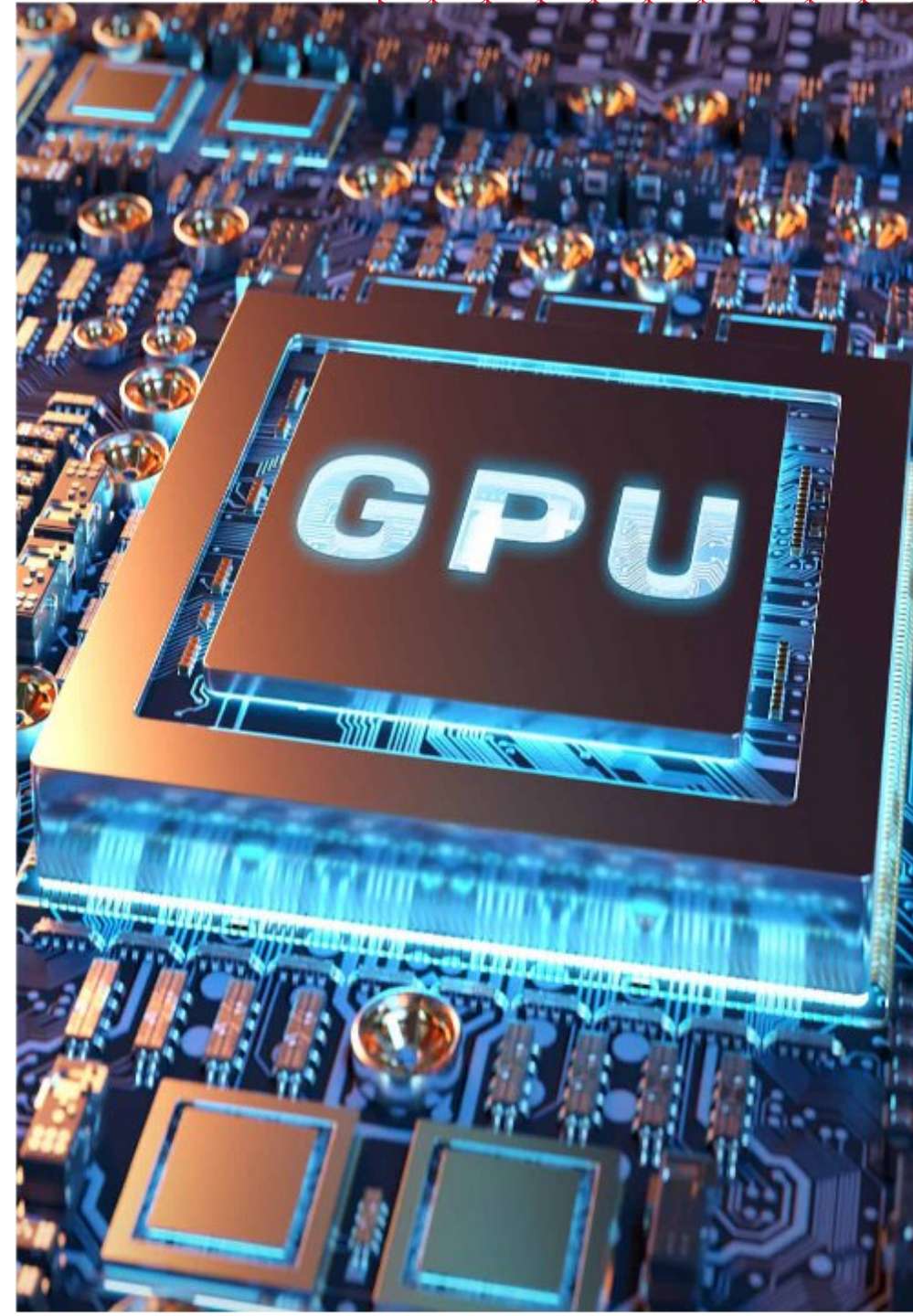




CExA, one year working with Kokkos at CEA

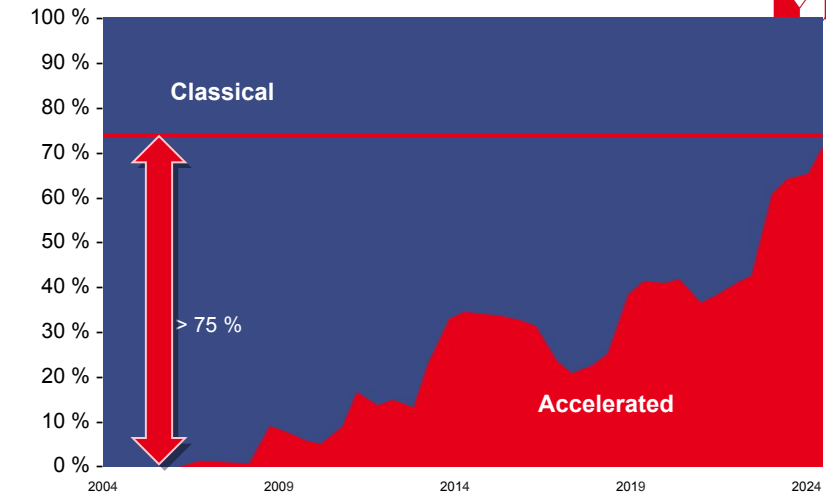
2025-10-25

Julien Bigot & the CExA team

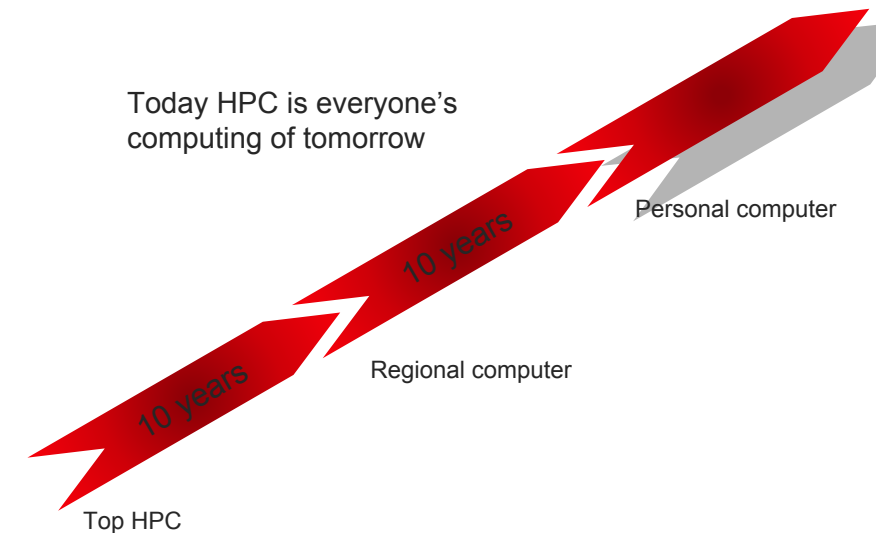


Context

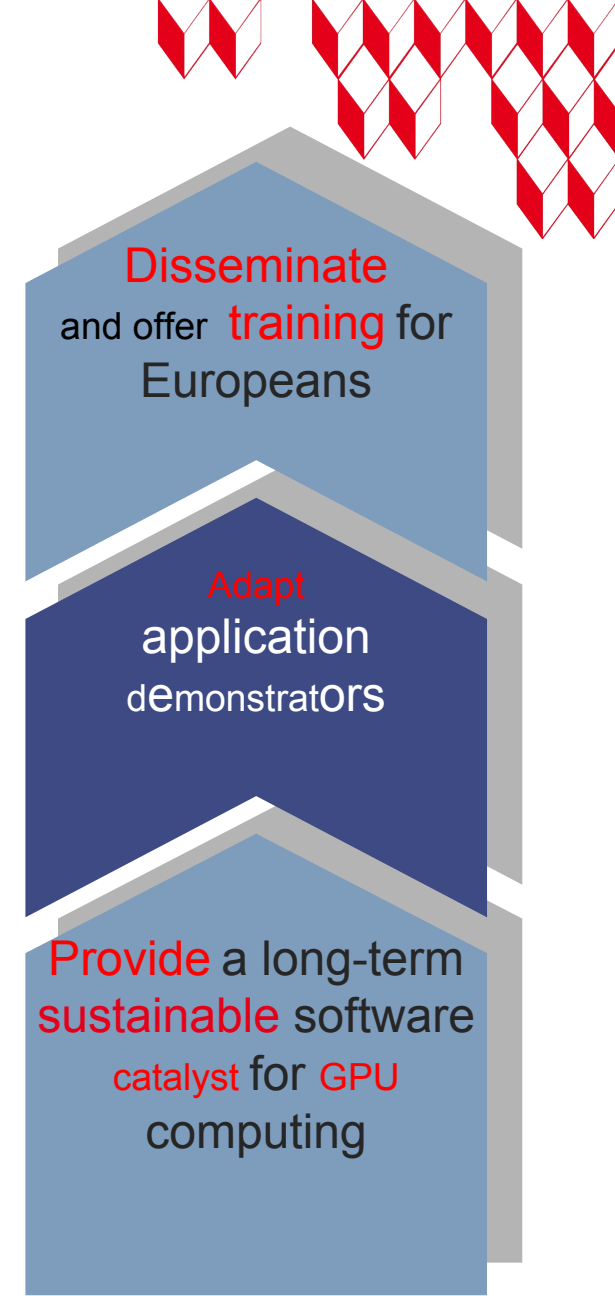
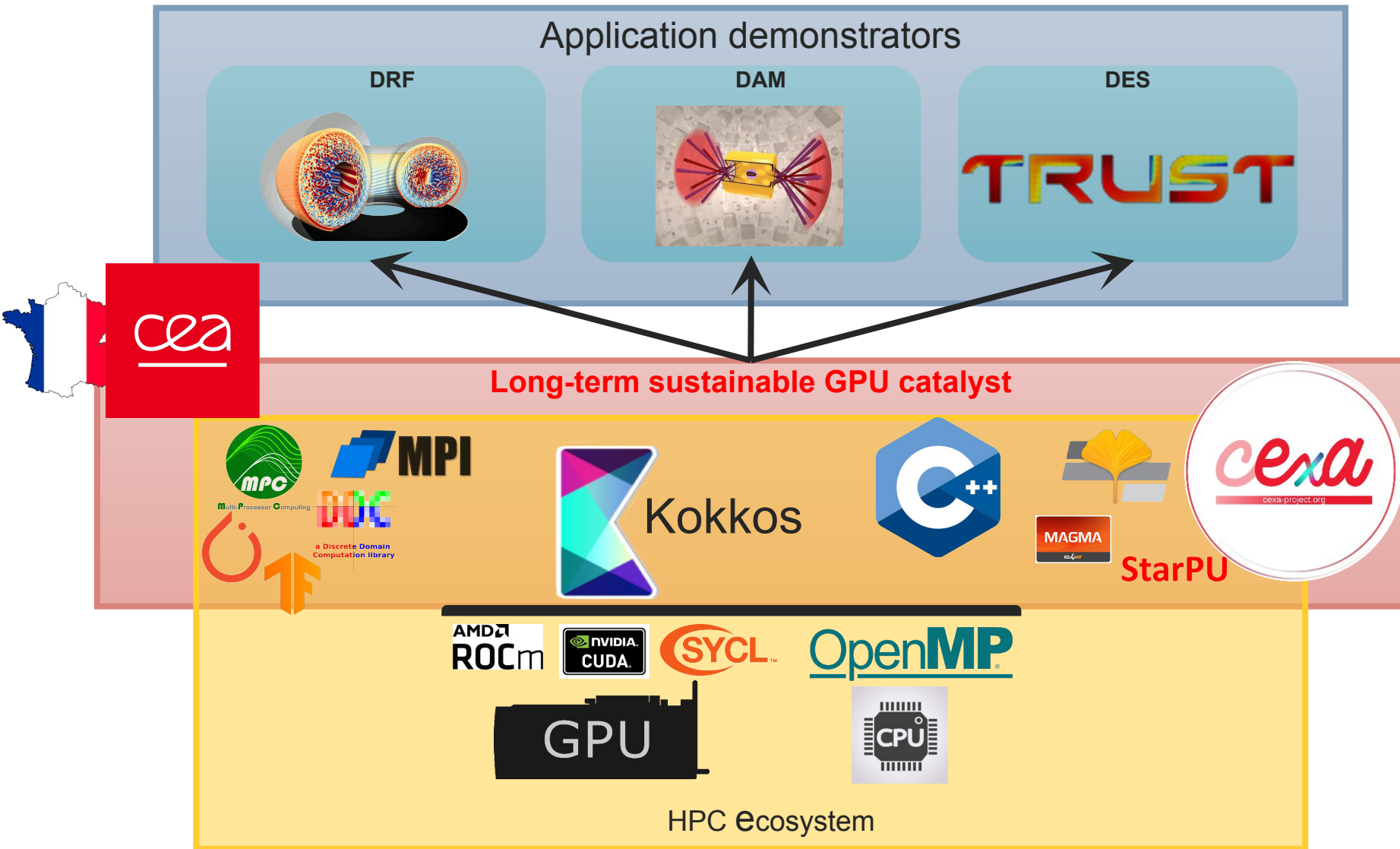
- HPC is a tool in a wide range of domains, **source of competitiveness**
 - At CEA, we host machines
 - To take part in the French & European HPC ecosystem
- We just entered the **Exascale** era, that means **GPU**
 - European pre-Exascale systems: **Mix of AMD & Nvidia**
 - First Exascale machines planned in Europe for 2024/2025
 - Jupiter machine at Jülich (Germany) => Nvidia & Rhea
 - **Alice Recoque** machine at **CEA/TGCC** (open call)
 - Need to re-develop applications with **Performance portability**
- GPU programming models: **software catalysts**
 - France and Europe had great research but no production tool
- A need for a **long-term sustainable solution**
 - Adapted to **our hardware and software specificities**
 - **Trust** in the roadmap



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



CExA project: goals



How to generate code a GPU can run?

- 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

GPU transparency

Performance

Performance portability

Domain abstractions

Generality

How to generate code a GPU can run?

- 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

GPU transparency

Performance

Performance portability

Domain abstractions

Generality

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**

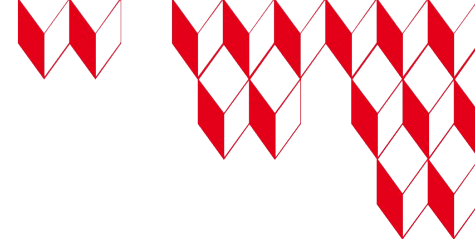
Available solutions

- 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

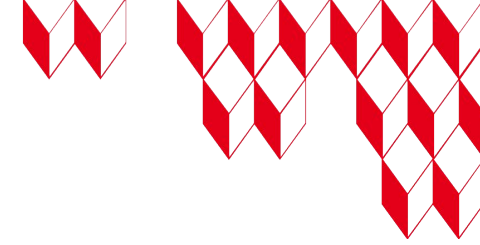
```
View<double*, Kokkos::HostSpace> x(Ni);
View<double*, Kokkos::HostSpace> y(Nj);
View<double*> A(Nj, Ni);

{
    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

Kokkos

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

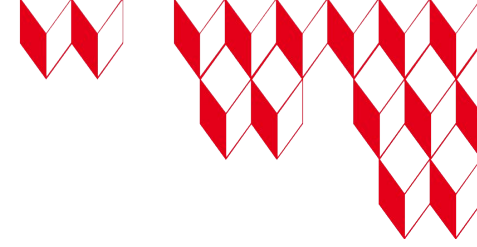
Available solutions

- 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
 - Requires to re-design applications for GPU
- **Library**
 - Suited to language with deep **encapsulation**: C++, ...
 - On top of vendor backends: easier to port to **new hardware**
 - Requires to re-write applications for GPU

What's in Kokkos



Multi-dimensional arrays

- Layout auto change for performance

Other containers

- Key-value maps, ...

Automatic ref-counted Host/Device memory allocation & management

Host/device memory transfers

Support of “dual” arrays with one version on each side

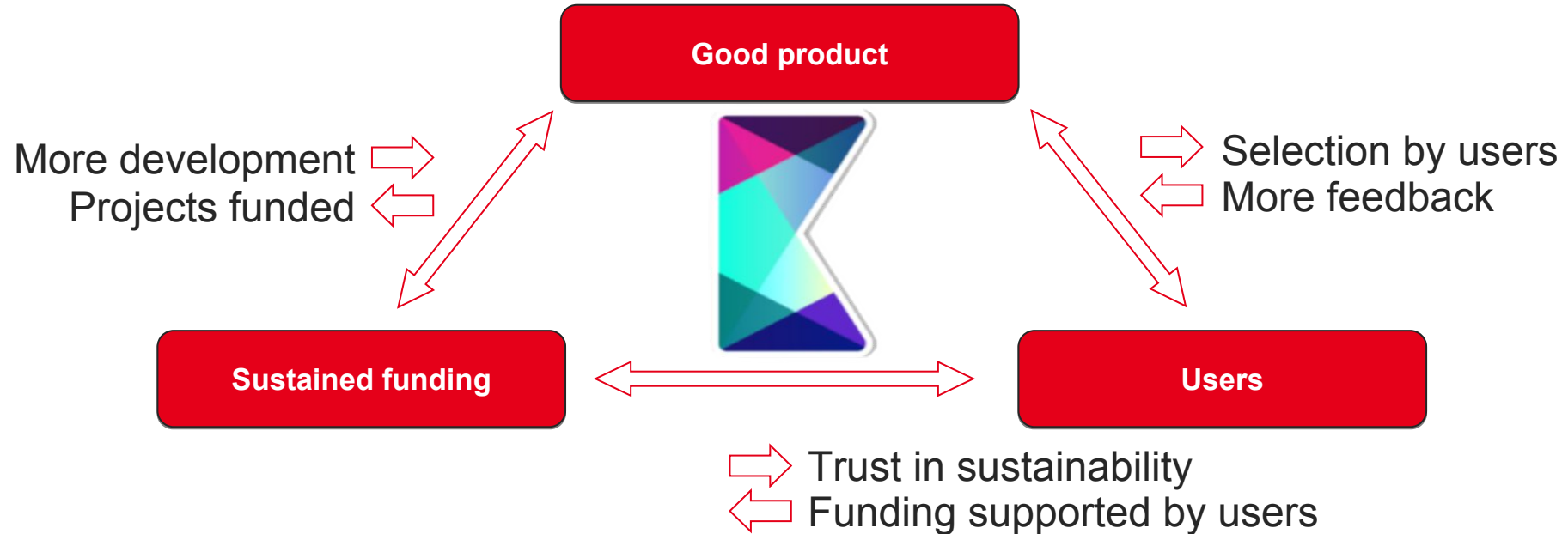
- Up-to-date tracking & automatic transfers when required

Scratch memory

- Using “core-local” fast memory on the device

- Parallel patterns w. asynchronous support
 - Independent interactions, Reductions, Scans
- Iteration strategies
 - Tiled, Hierarchical, ...
- Algorithms
 - Sorting
 - Random number generation
 - Most of STL parallel algorithms
 - ...
- QoL features: portable printf, etc.
- Portable atomic operations
- SIMD
- Coarse & fine-grain tasks
- And much more...

Kokkos at the center of a virtuous cycle



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

© Christian Trott & Damien Lebrun Grandie

Kokkos an anteroom for standard C++

C++ is (at last) standardizing base tools for HPC

- **Parallel programming** is slowly entering the **ISO C++ language**
 - Parallel algorithms, sender/receivers, etc.
- **The Kokkos team** leads the **standardization** of many required features
 - Multi-D arrays (`std::mdspan`)
 - Vectorization (`std::simd`)
 - Linear algebra (`std::linalg`)
 - And much more to come (mixed precision, etc.)

Kokkos offers a **stable API** today for the features of the **C++ of tomorrow**

- Standardization is slow (9 years for `mdspan`)
 - Consensus with all communities
- Kokkos offers the **features today**
 - And will **keep maintaining the API on top of standardized ISO C++**
 - With added interoperability layers (Cf. `kokkos::view` / `std::mdspan`)
 - And in a **GPU-compatible** implementation (Cf. `kokkos::array`)





HIGH PERFORMANCE SOFTWARE FOUNDATION

HPSF Goals

- Provide neutral home for key HPC projects to enable collaboration between government, industry and academia
- Promote use of HPSF projects
- Ensure that HPC software is accessible and reliable by providing CI and turn-key builds
- Ensure that HPC software is secure and ready for cloud through collaborations with CNCF and OpenSSF
- Sponsor events and training to grow a diverse, skilled workforce for software in the HPSF ecosystem.

Members

Premier



General



Associate



CExA made its choice

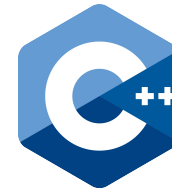
“adopt and adapt” strategy based on Kokkos

- 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++**



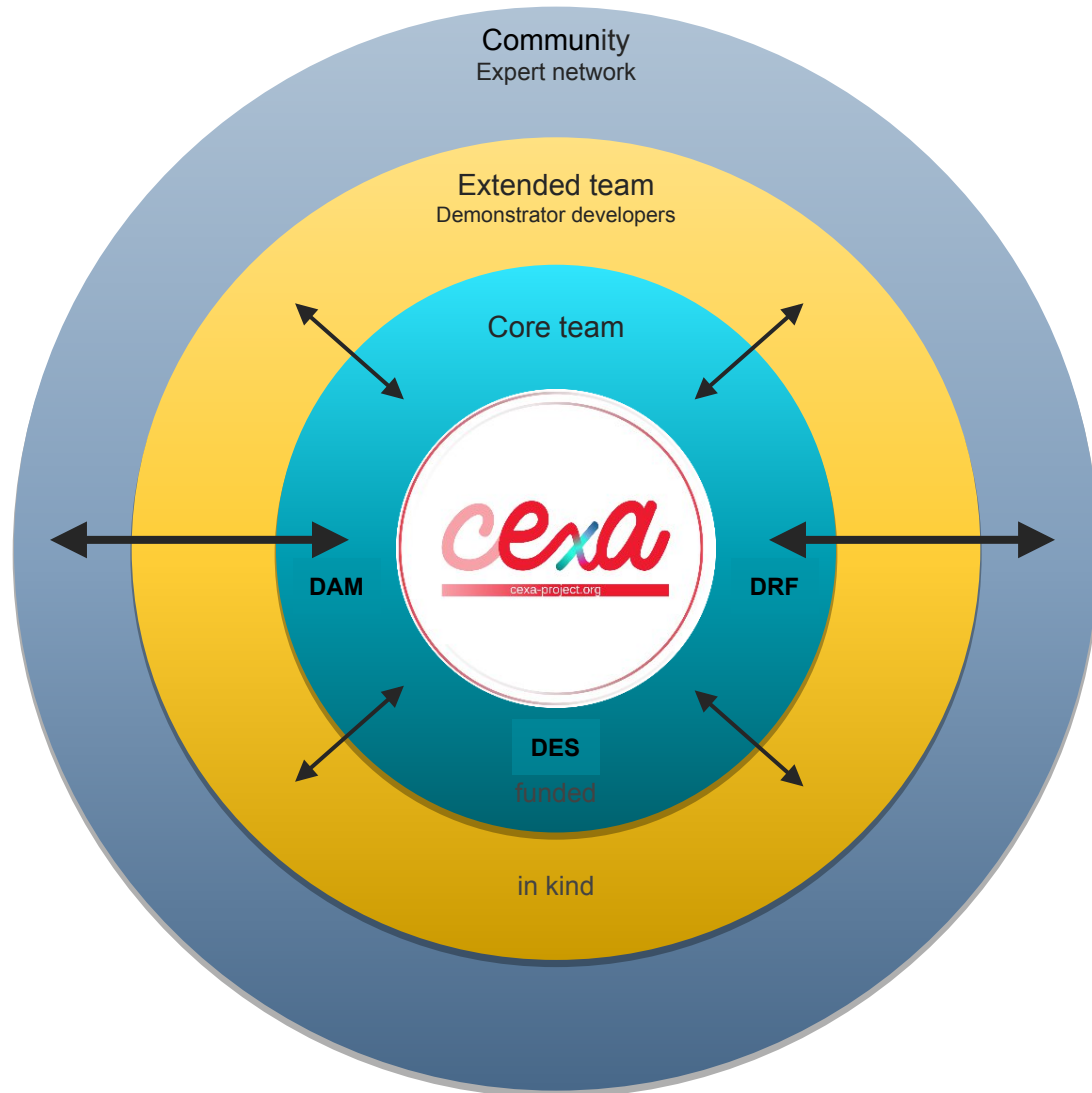
- A **stepping stone** one step ahead toward **parallel C++**



- 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
- 8 permanent researchers from all over CEA
- 3 recruitments done, 5 more funded
 - 1 as a permanent researcher !
- Funding for 2 or 3 more hires expected next year

■ Extended team

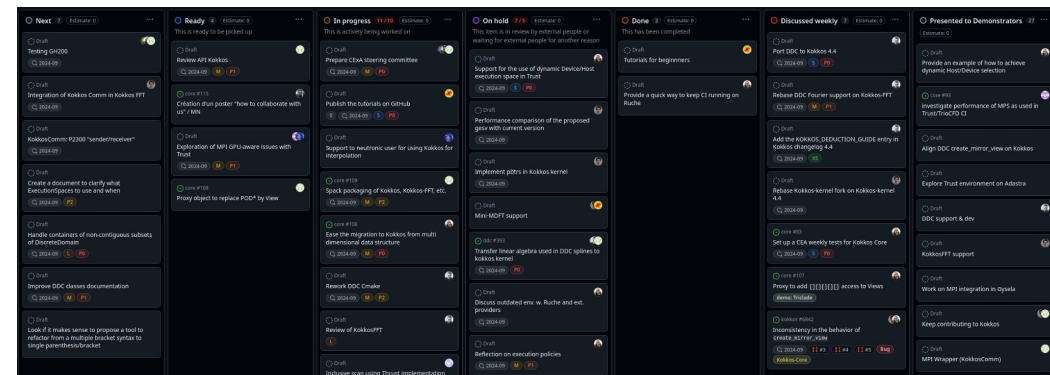
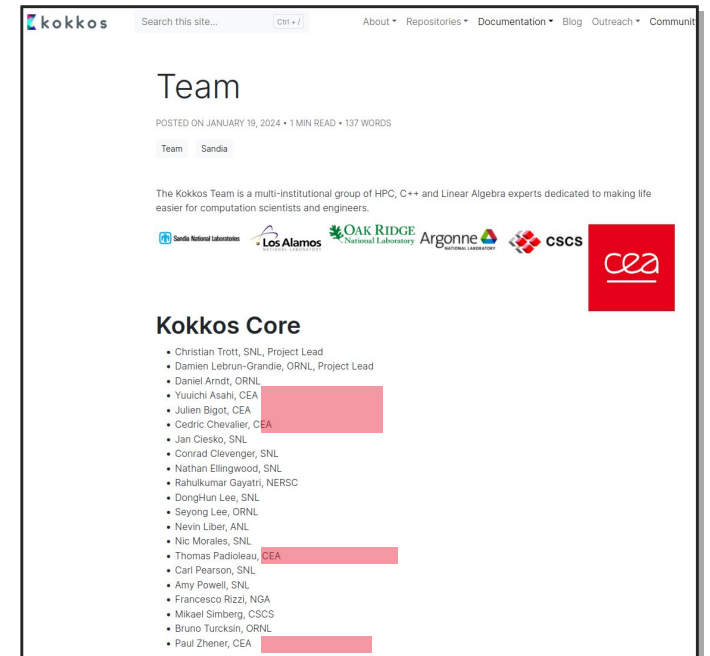
- Demonstrator developers
 - Not funded
 - Find their own interest in the participation
- 2-3 new demonstrators every year

■ Community

- Federation of an expert network
- Co-design of CExA:
 - Identification of needs
 - Usage of CExA in applications
- Priority target for dissemination
- Sustainability of the work

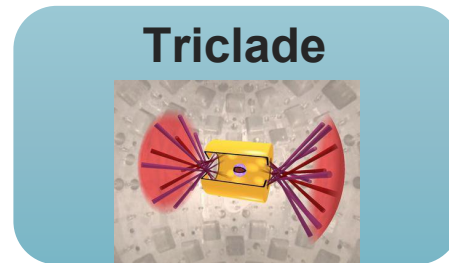
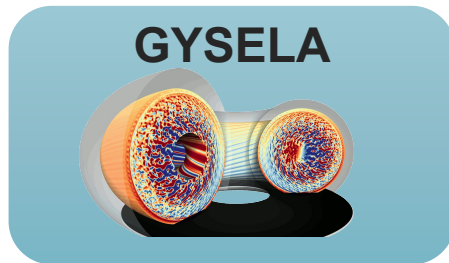
How we work

- CExA core team is distributed (Saclay, Digiteo, Bruyère le Chatel)
 - We rely on remote collaboration tools
 - But we take one full day a week together in our dedicated office
- CExA is now a core contributor to the international Kokkos team
 - With the end of ECP, more than a third of the manpower
 - 2 weekly meetings with our international colleagues
- We adopt an agile approach
 - Our developments are guided by the needs of demonstrators
 - Priorities are given once a month by them
 - Monthly reporting to them
- We also dedicate some time to long-term goals
 - Training & dissemination
 - Code quality, core libraries improvement

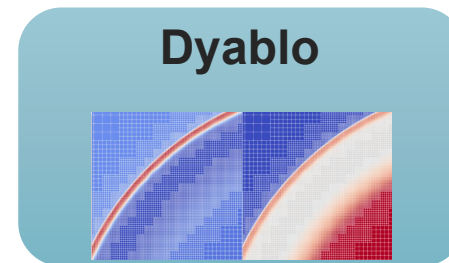


Our demonstrators

- 2023:



- 2024:

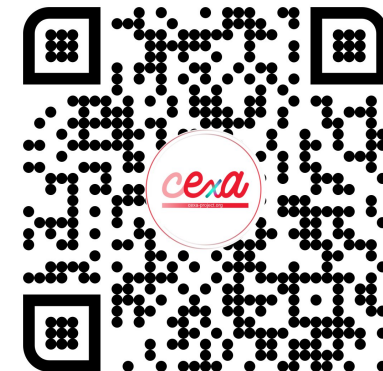
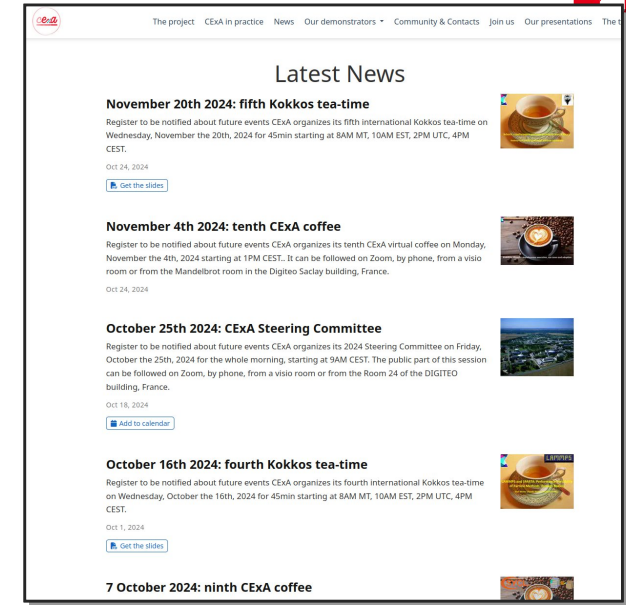


- and also ExaTract, MDFT, ...

- 2025: ???

Kokkos training & community animation

- Many Kokkos trainings
 - September 2023 with C. Trott & D. Lebrun Grandié in Saclay
 - March 2025 Hackathon at IDRIS
 - June 2024 w. D. Lebrun Grandié & L. Berger-Vergiat
 - November 2024 Mission Numérique CEA in Grenoble
 - January 2025 CEA/Riken winter school in Barcelona
 - January 2025 Hackathon w.
 - February 2025 Mission numérique in Cadarache
 - Summer school 2025 w. EDF & Inria
- Kokkos slack now has a `#general-fr` channel (~10% of the whole community)
- CExA virtual café once a month
 - Informal discussions, in French about Kokkos, its ecosystem & GPU at large
- Kokkos virtual tea-time once a month
 - Informal presentations, in English about Kokkos, its ecosystem & GPU at large
 - With our US partners



CExA: what's going on?



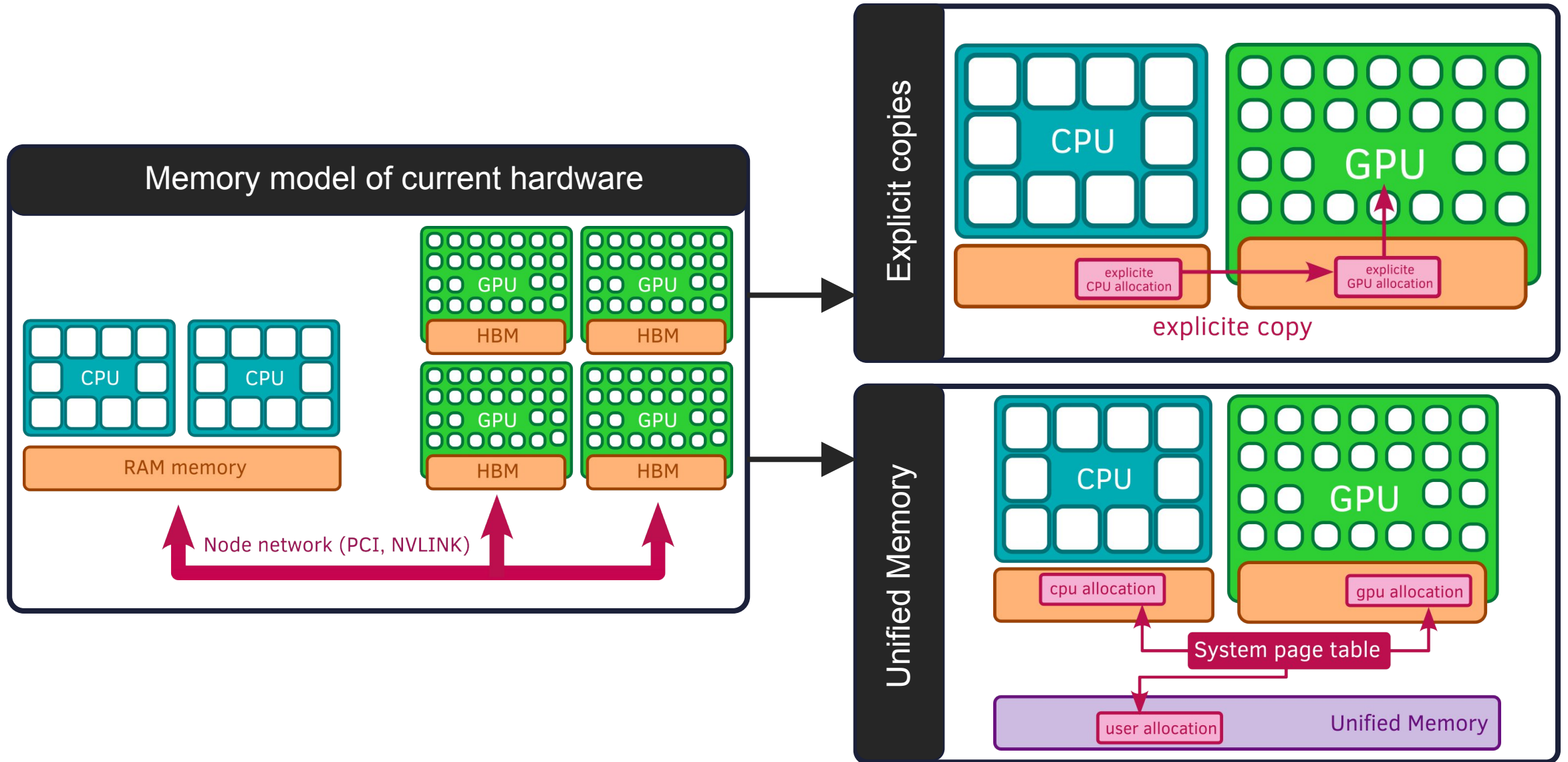
- Help with documentation
 - Website improvement
 - Cheat-sheets creation
- Support our applications
 - Test UVM viability & performance
 - Add required solvers to Kokkos-kernels
- Improve software quality
 - Setup GPU CI for CEA libraries
 - Maintaining Kokkos Spack recipes
 - Huge refactor & redesign of `create_mirror[_view][_and_copy]`
- Test hardware & improve kokkos for it
 - Intel PVC backend improvement
 - NVidia Grace Hopper memory management
- DDC
 - Discrete data & computation
 - Zero overhead discretization support
 - Named dimensions ala xarray
- Kokkos-FFT
 - Performance portable FFT
 - with a Kokkos API
- Kokkos-comm
 - Integrate Kokkos for Multi-node
 - Ease MPI usage
 - Rely on Kokkos metadata
 - Device-direct MPI
- Help with transition to Kokkos
 - From Fortran, OpenMP, plain C, ...



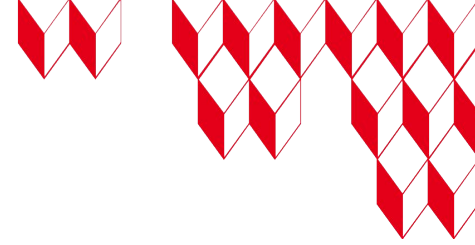
2 ■ Memory performance

Mathieu Lobet

Memory management is crucial in heterogeneous computing



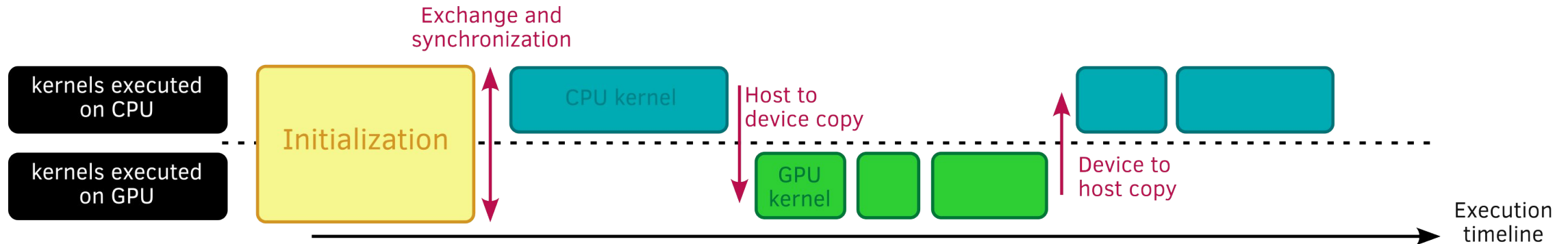
Challenges regarding the memory model



TRUST

Questions for our demonstrators:

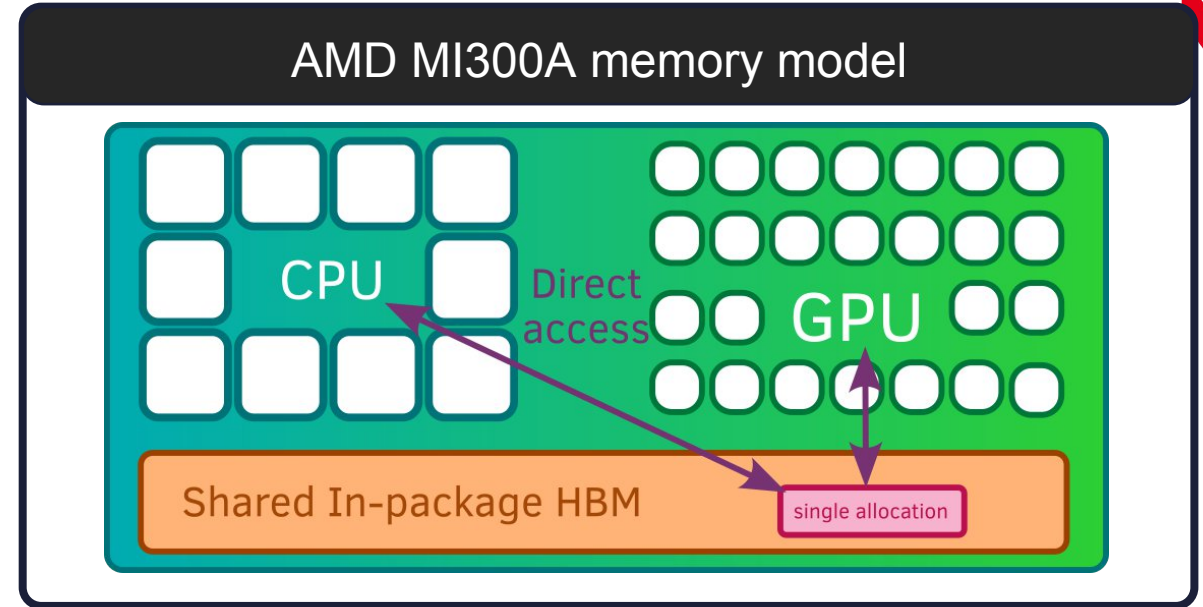
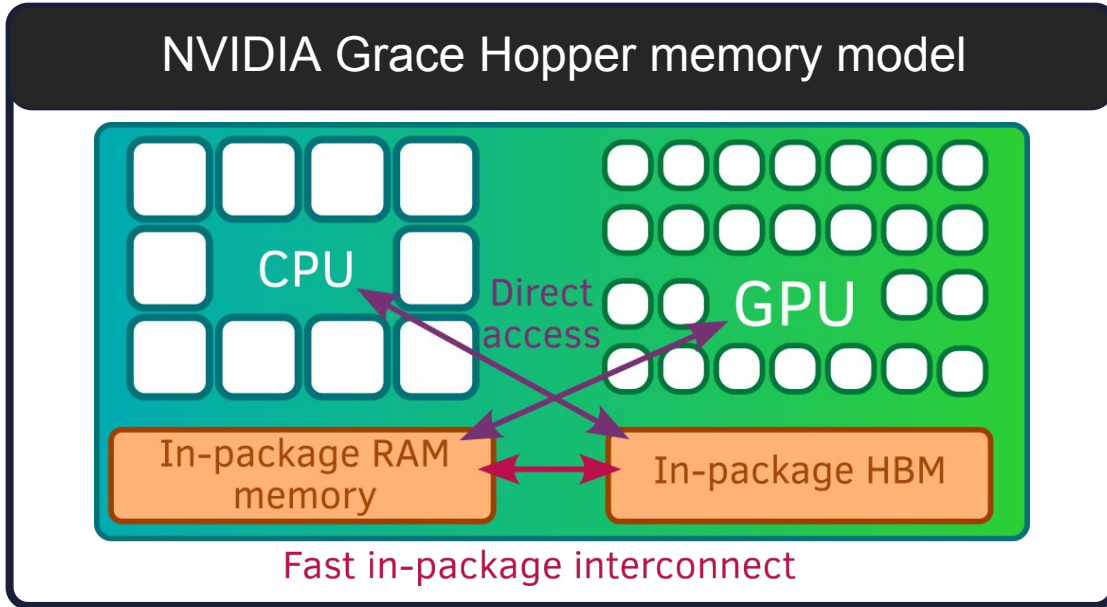
- Exploration of the memory management for Trust



For all CEA applications :

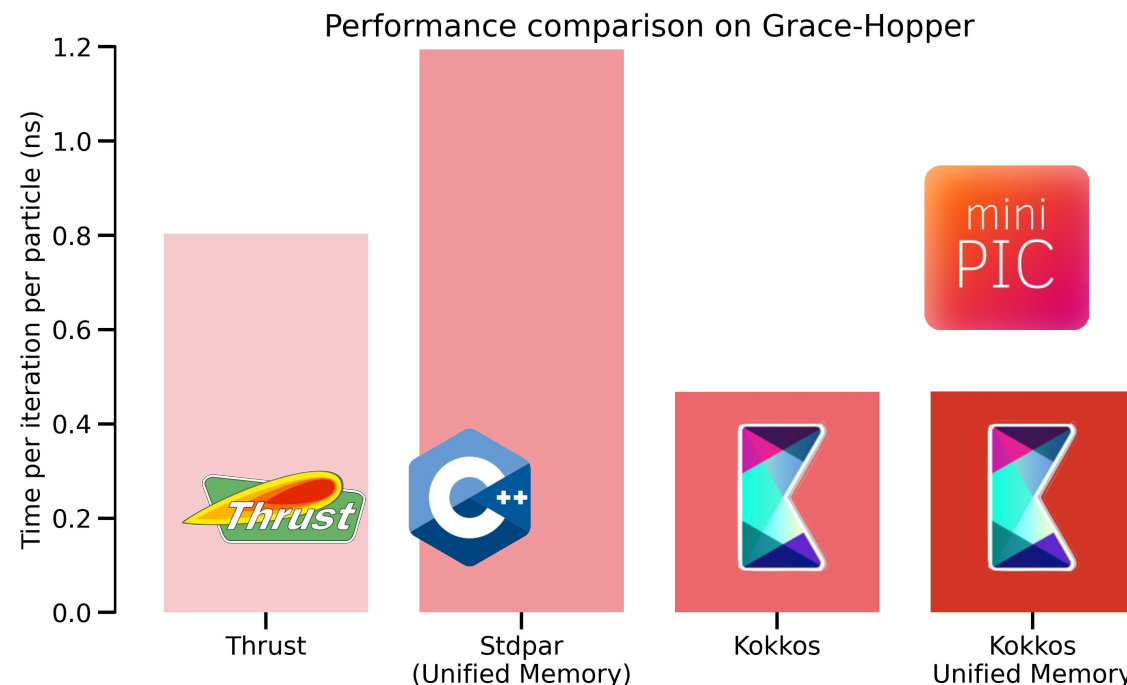
- How to facilitate the transition period when porting step by step a CPU code to GPU?
- What is the best strategy regarding performance, portability, productivity and evolutivity?
- What is the best strategy for real world usage: outputs, loose-coupling, in-situ, etc?
- Advanced optimization like asynchronism?

New challenges with coming architectures



Exploration for anticipating the hardware evolution

- Extensive experiments with the different memory models from Kokkos
 - Using explicit vs “shared” vs “direct” memory management (on Intel PVC and Nvidia Grace-Hopper)
 - Use of Kokkos tests, benchmarks and mini-apps for complete evaluation
- Contributions to Kokkos support
 - Optional Grace-Hopper Kokkos option is to use “cudaMallocManaged” as it allows better GPU performance
 - Support MI300A since Kokkos 4.2.01
 - Specific support for Grace-Hopper, released in Kokkos 4.4.01
- **Conclusion: benefit of using Kokkos**
 - Facilitates the memory management and offers flexibility
 - Prepared to forthcoming technologies thanks to our early access evaluation
 - Decent performance compared to other programming models using simple (non-optimized) kernels





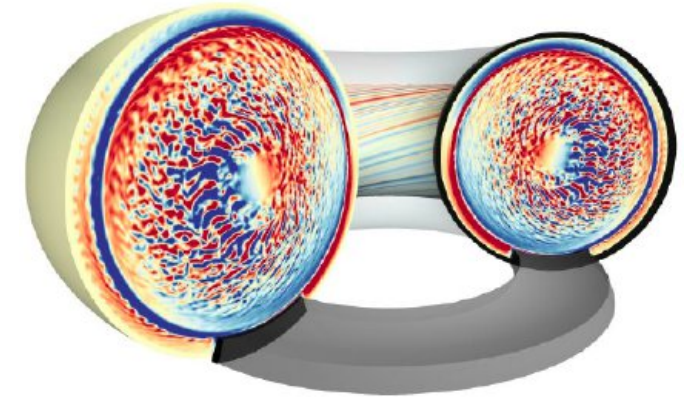
3 ■ Kokkos-FFT

Yuuichi Asahi

Why KokkosFFT, who needs that?

- Using Kokkos to port a legacy application which relies on FFT libraries
 - Fluid simulation codes with periodic boundaries, Plasma turbulence, etc
 - BiGDFT (Fortran), MDFT (Fortran), GYSELA-X (C++), etc
- Having a Kokkos code and willing to integrate in-situ data processing with FFTs
 - Spectral analyses, Low path filter, etc
- Not willing to get through documentations of de facto standard FFT libraries
 - Benefit from powerful FFT libraries as simple as **numpy.fft**

GYSELA-X (plasma turbulence)
Periodic along toroidal direction



kokkos-fft

Non-uniform interfaces for vendor libraries

FFTW

```
plan = fftw_plan_many_dft(  
    rank, fft_extents.data(), howmany,  
    idata, in_extents.data(), istride, idist,  
    odata, out_extents.data(), ostride, odist,  
    sign, FFTW_ESTIMATE);
```

ROCFFT

```
rocfft_plan_create(&plan, place, fft_direction, precision,  
    fft_extents.size(),  
    fft_extents.data(),  
    howmany,  
    description);
```

CUFFT

```
cufftPlanMany(&plan, rank, fft_extents.data(),  
    in_extents.data(), istride, idist,  
    out_extents.data(), ostride, odist,  
    type, howmany);
```

oneMKL

```
plan = std::make_unique<PlanType>(fft_extents);  
plan->set_value(oneapi::mkl::dft::config_param::INPUT_STRIDES,  
    in_strides.data());  
plan->set_value(oneapi::mkl::dft::config_param::OUTPUT_STRIDES,  
    out_strides.data());  
  
// Configuration for batched plan  
plan->set_value(oneapi::mkl::dft::config_param::FWD_DISTANCE, idist);  
plan->set_value(oneapi::mkl::dft::config_param::BWD_DISTANCE, odist);  
plan->set_value(oneapi::mkl::dft::config_param::NUMBER_OF_TRANSFORMS,  
    static_cast<std::int64_t>(howmany));  
  
// Data layout in conjugate-even domain  
int placement = is_inplace ? DFTI_INPLACE : DFTI_NOT_INPLACE;  
plan->set_value(oneapi::mkl::dft::config_param::PLACEMENT, placement);  
plan->set_value(oneapi::mkl::dft::config_param::CONJUGATE_EVEN_STORAGE,  
    DFTI_COMPLEX_COMPLEX);  
  
sycl::queue q = exec_space.sycl_queue();  
plan->commit(q);
```

numpy.fft

```
xr2c_hat = np.rfft(xr2c, axis=-1)
```

kokkos-fft

```
KokkosFFT::rfft(exec, xr2c, xr2c_hat, /*axis=*/-1);
```

Key features of Kokkos-fft



As simple as `numpy.fft`, as fast as vendor libraries

- 1D, 2D, 3D standard and real Fast Fourier Transforms over 1D to 8D Kokkos Views
 - Batched plans are automatically used if View Dim > FFT Dim
- Simple interfaces like **`numpy.fft`** (out-of-place and in-place)
 - View is all we need: No need to access the complicated FFT APIs
 - Much safer APIs: No need to use raw pointers directly, compile time/runtime errors for invalid usage
- Supporting multiple CPU and GPU backends (FFTs are executed on the stream/queue used in the Execution space)
 - **`SERIAL`**, **`THREADS`**, **`OPENMP`**, **`CUDA`**, **`HIP`** and **`SYCL`**
 - FFT libraries dedicated to Kokkos backends are automatically enabled
- Supported data types: float, double and `Kokkos::complex`
 - Limited to contiguous layout only: **`LayoutLeft`** and **`LayoutRight`**
 - **`DefaultExecutionSpace`** and **`DefaultHostExecutionSpace`** supported

APIs (numpy.fft + FFT Plan)

```
constexpr int n0 = 128, n1 = 128, n2 = 16;

// 1D batched R2C FFT
View3D<double> xr2c("xr2c", n0, n1, n2);
Kokkos::Random_XorShift64_Pool<> random_pool(12345);
Kokkos::fill_random(xr2c, random_pool, 1);

View3D<Kokkos::complex<double>> xr2c_hat("xr2c_hat", n0, n1, n2/2+1);
KokkosFFT::rfft(exec_space(), xr2c, xr2c_hat, KokkosFFT::Normalization::Backward, -1);
```



```
import numpy as np

n0, n1, n2 = 128, 128, 16

# 1D batched R2C FFT
xr2c = np.random.rand(n0, n1, n2)

xr2c_hat = np.fft.rfft(xr2c, axis=-1)
```



APIs

- KokkosFFT::<func> equivalent to numpy.fft.<func>
- Namespaces: KokkosFFT (APIs) and KokkosFFT::Impl (implementation details)
- Macros: KOKKOSFFT_*

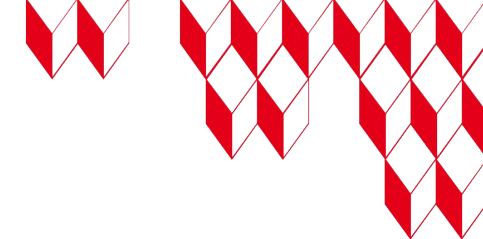
Implementations

- Internally, (maybe) transpose + FFT plan creation + FFT execution + (maybe) normalization
- Errors if there is inconsistency between exec-space and Views
- FFT plans can be reused (important for cufft and rocfft)

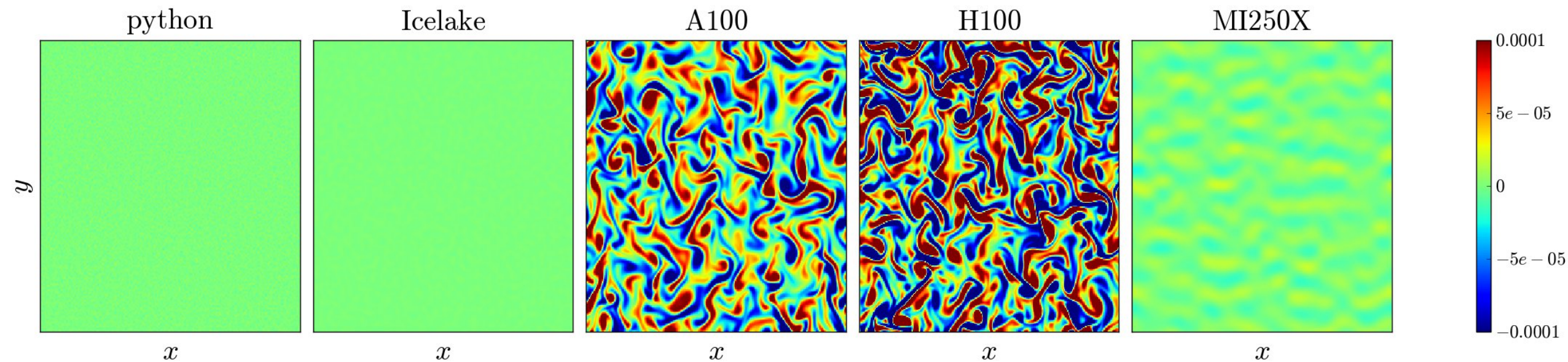
KokkosFFT documentation page showing navigation links: Getting started, Finding FFT libraries by CMake, API Reference, Examples. Below is a MongoDB advertisement: "Develop and launch modern apps with MongoDB Atlas, a resilient data platform. Ad by EthicalAds".

KokkosFFT documentation page showing code examples. The first example is C++ code for a 1D real to complex transform. The second example is a Python script using numpy.fft.rfft. A note states: "It is assumed that backend FFT libraries are appropriately installed on the system." Below the note is a table of contents with links to Getting started, Quickstart guide, Building KokkosFFT, and Using KokkosFFT.

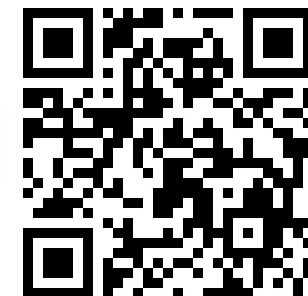
Kokkos-fft: As simple as numpy as fast as vendor libraries



Experiment in 2D Hasegawa-Wakatani solver relying on convolution with FFT
(1024 x 1024), 100 iterations (w/o I/O), **snapshot after 0.13 [s]**



Device	Icelake (python)	Icelake (36 cores)	A100	H100	MI250X (1 GCD)
LOC	232	508	508	508	508
GB/s	205	205	1555	3350	1600
Elapsed time [s]	834	9.2	0.25	0.13	0.41
Speed up	x 1	x 90.7	x 3336	x 6182	x 2034





4 ■ Packaging with Spack

Cédric Chevalier

From Performance Portability to Production Availability



Goal: Install and Use our Kokkos enabled applications on production clusters

Needs:

- Ease the build of complex software stack on various computers
- Build optimized applications to deliver performance to the end user

Constraints:

- Every computing center has its way to do things
- We want to keep knowledge of how our applications are built
- We want to build automatically

Software Deployment with Spack

Spack is an open-source (HPSF) package manager software to install HPC software stack on leading class computers

Spack works using **recipes**, that describe how to build a package

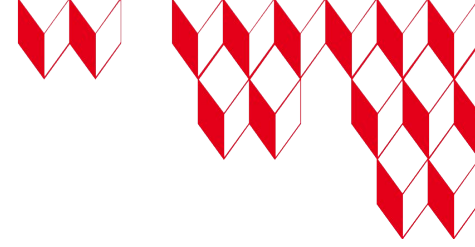
It contains:

- How to get the source code of the package
- What are the dependencies of the package
- What configuration/build options can we choose
- How to really build the package

Spack will solve the dependency hell problem and choose a feasible global configuration, setting compatible options across packages



Kokkos and Spack



Spack has a Kokkos that is used by all packages that rely on Kokkos (eg. LAMMPS, portage, PETSc, Trilinos, ...)

Spack is open-source and anyone can contribute to recipes, but each package has a maintainer to help validate the changes

Since March, we are also maintainer of the Kokkos' Spack recipe:

- We are on the front line to gather issues on new hardware, and that helps us to ensure portability
- We control the changes in the recipe and we do not just endure them

On going work

Add new packages around Kokkos

- Kokkos-FFT
- Kokkos-Comm



It will

- Improve the deployment to end-users, some stack deployments are already using Spack
- Allow our users to use it more easily on any kind of computers, even the one we do not have access to
- Broaden the user community, making us to be aware earlier of issues (if any!)





5 ■ Continuous integration (CI)

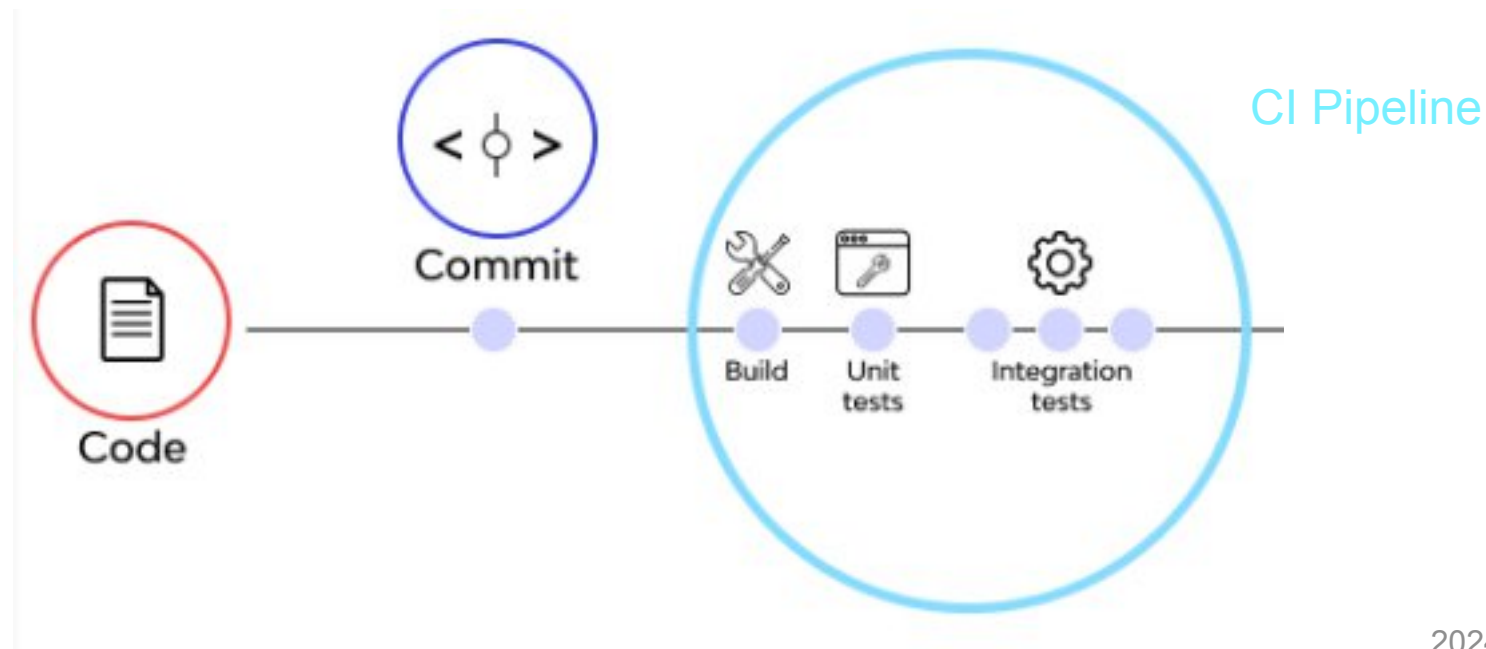
Rémi Baron

CI needs for CExA

CI is a key ingredient to ensure that a software project can accept frequent source code contributions while ensuring to stay in a workable state.

At least two reasons to have a continuous integration inside CExA :

- To help integration of contributions directly inside Kokkos on our hardware
- To verify our own production ready libraries like the Kokkos-FFT/Kokkos-Comm, DDC

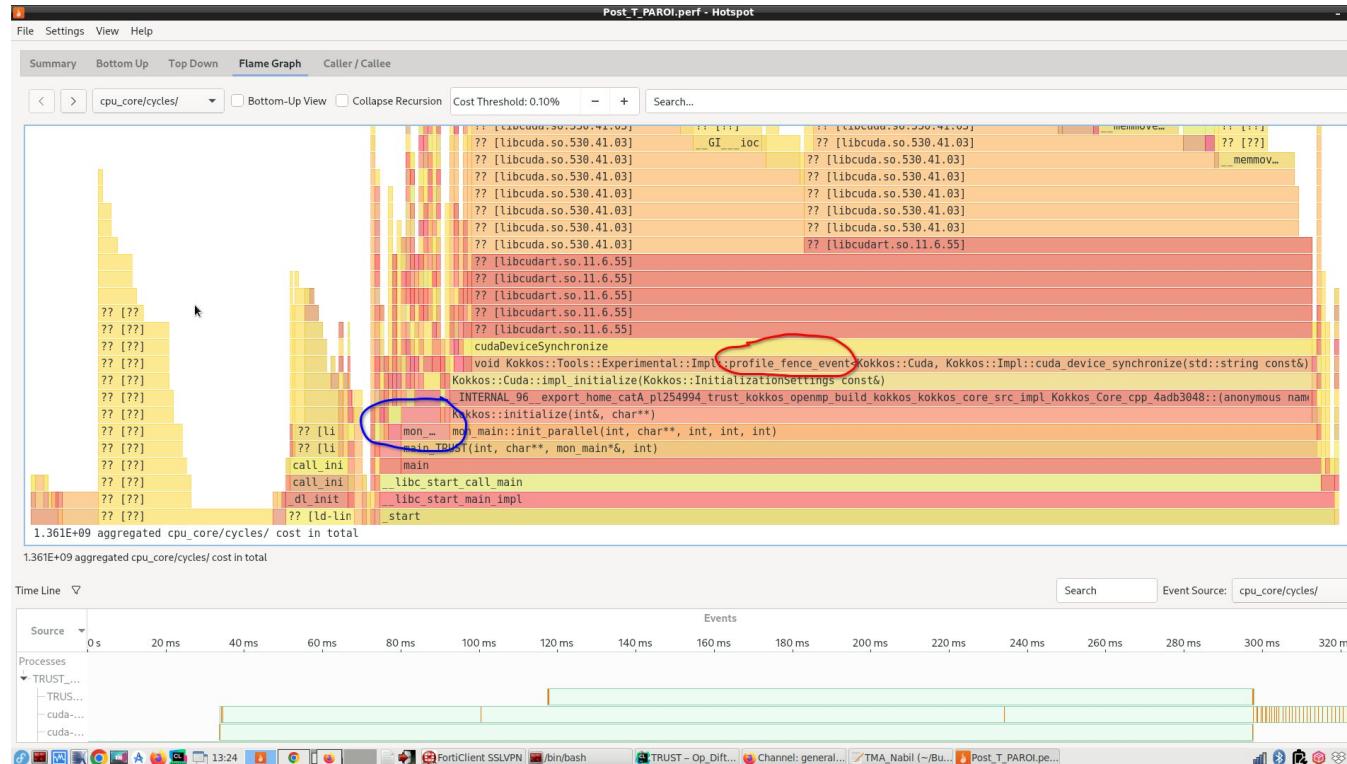


CI of demonstrators

During one monthly meeting with our demonstrators, the TRUST team reported that their CI was slower than expected on GPU.

Some very short unit tests when executed on CPU were much slower when executed on GPU.

Kokkos initialization functions seemed to be the origin of the overhead.



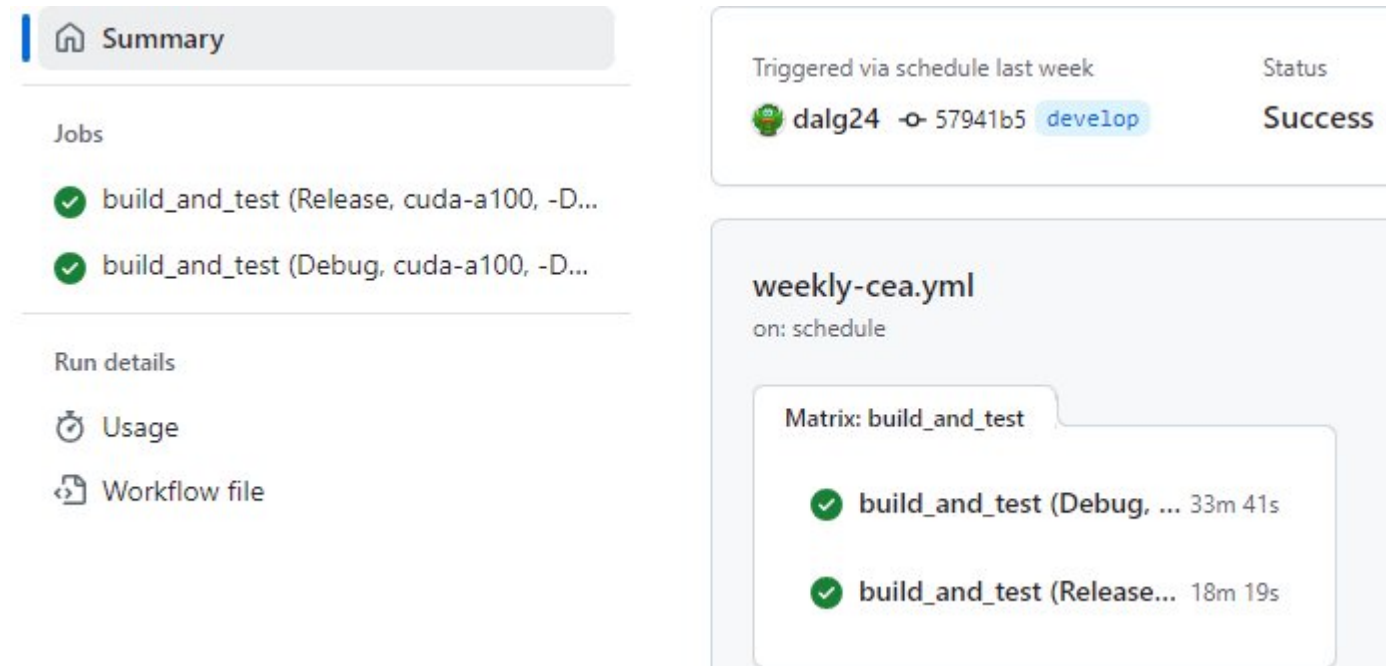
CExA's solution

As the source code of Kokkos is hosted on Github, it was decided to set up a GitHub Actions Runner on **Ruche** cluster of Paris-Saclay Mesocentre.

The current implementation uses a self-hosted runner on a login node of the cluster using a dedicated login. With the help of administrators of the Kokkos project a **test workflow** was added.

It runs weekly to test on A100 cards, with release and debug builds.

SLURM jobs are submitted and results are reported to Github.



The screenshot shows a GitHub Actions workflow run summary. On the left, a sidebar contains navigation links: 'Summary' (selected), 'Jobs', 'Run details', 'Usage', and 'Workflow file'. The 'Jobs' section lists two successful jobs: 'build_and_test (Release, cuda-a100, -D...)' and 'build_and_test (Debug, cuda-a100, -D...)', both marked with green checkmarks. The main content area shows the workflow 'weekly-cea.yml' triggered via schedule last week, with a status of 'Success'. Below this, a matrix for 'build_and_test' is shown, listing two successful jobs: 'build_and_test (Debug, ... 33m 41s)' and 'build_and_test (Release... 18m 19s)', both marked with green checkmarks.

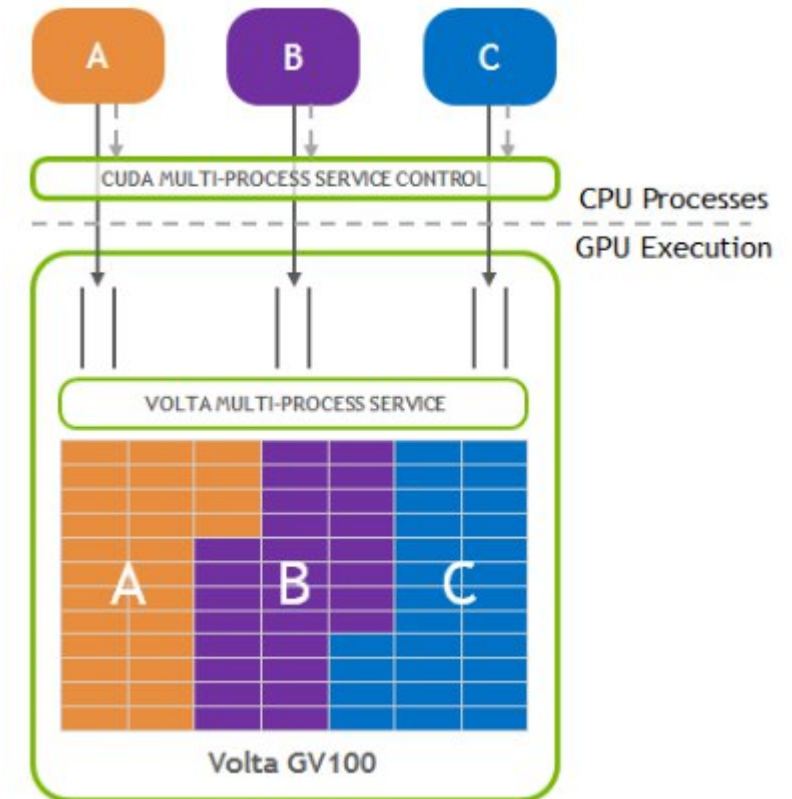
CExA's support of TRUST CI

Initial actions done :

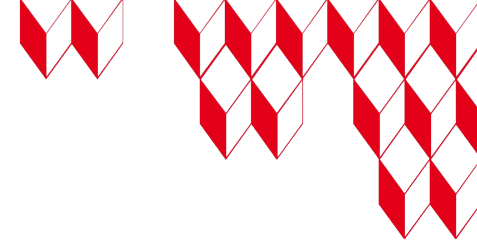
- Measurements were done to assess the overhead introduced by Kokkos initialization functions.
- Investigations were made to see if it can be decreased.

After this a more **global investigation** of the CI process was made :

- TRUST CI uses CTest to launch the tests and a Multi-Process Service (MPS) daemon so that several tests can use the same GPU card.
- Suggestions were made to configure the GPU in a special mode as recommended in the MPS documentation.
- TRUST team also fixed a memory allocation bug on the device which also helped.



Current status and future ideas



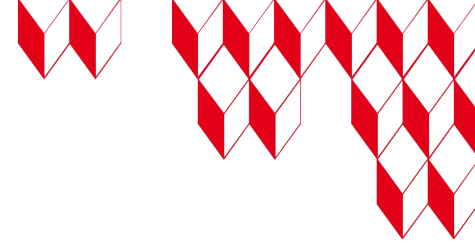
- We have a **working CI** for the needs of CExA on our own hardware.
- There is on-going work to improve this CI implementation in order to use Singularity **containers**. This may be useful to codes that want to run a CI on cluster with GPUs.
- Ideas to improve **Kokkos CI** itself : for instance with a common CDash dashboard to collect all tests results in a more unified way.
- We will continue to support any question regarding CI from our demonstrators.



6 ■ Support for code migration

Paul Zehner

Motivations

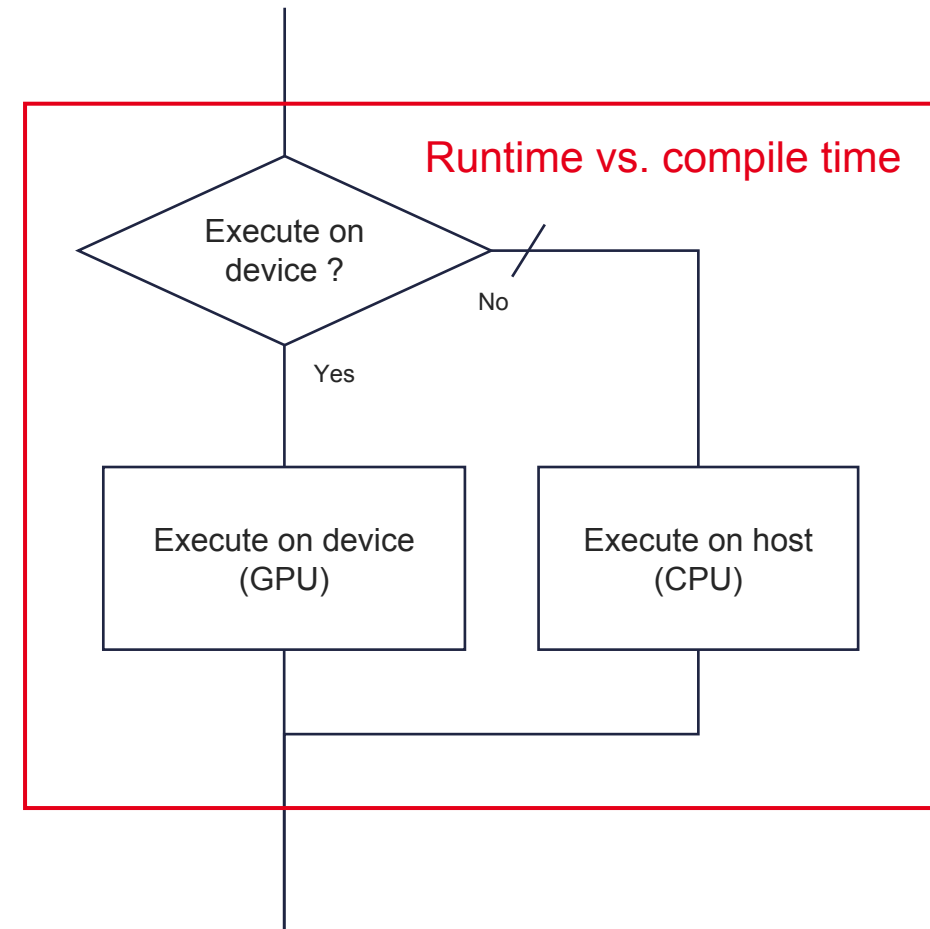


Demonstrators have needs for code migration, from old practices to newer ones

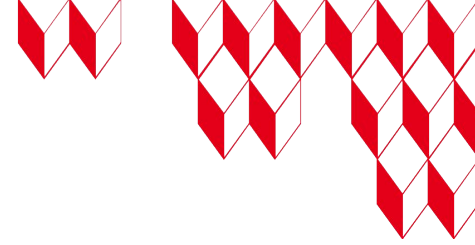
- **TRUST:** OpenMP Target
- **Triclade:** C++ with plain-old-data C arrays
- **BigDFT:** Fortran

The need of dynamic execution

- OpenMP Target allows to decide if the code is executed on the host or on the device **dynamically** with the `if` clause
- Kokkos only allows to decide this at **compile time**
- TRUST requires to migrate OpenMP Target code that uses dynamic scheduling for its internal classes
- This helps transitioning and can be useful for other codes



Development of dynamic execution library

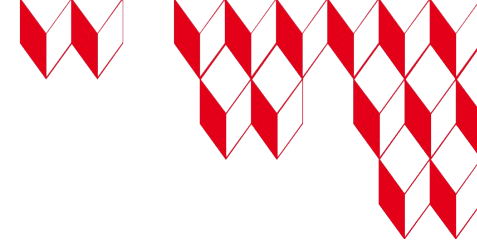


- Discussion in the Kokkos community
- Dynk helper (**D**ynamic **K**okkos)
- Allows programmers to use dynamic scheduling by either
 - Using a layer on top of Kokkos (difficult to maintain in the long run)
 - Wrapping their code (no full compiler support yet)
- Experimentally in use in TRUST

Backend	Compiler	Layer	Wrapper if	Wrapper 2 functions	Wrapper function	Wrapper functor	Wrapper lambda
Cuda	NVCC	✓	✓	✓	⚠	⚠	✗
Cuda	Clang	✓	✓	✓	⚠	⚠	⚠
HIP	ROCM	✓	✓	✓	⚠	⚠	⚠
SYCL	Intel	✓	✓	✓	⚠	⚠	⚠

- ✓ Works out of the box
- ⚠ Requires recent enough compiler (C++20)
- ✗ Not supported

The need to access new objects the old way



- Plain-old-data C arrays use the `[]` syntax to access their elements
- Kokkos provides views that use the `()` syntax to access their elements
- Triclade uses plain-old-data C arrays
- As a transition, the signature of functions is updated to use Kokkos views, but the body of the function remains unchanged
- This helps transitioning and can be useful for other codes

Multidimensional plain-old-data C arrays

```
myArray[i][j][k] = 10;
```

Multidimensional Kokkos views

```
myView(i, j, k) = 10;
```

Transition from plain-old-data C arrays to Kokkos views

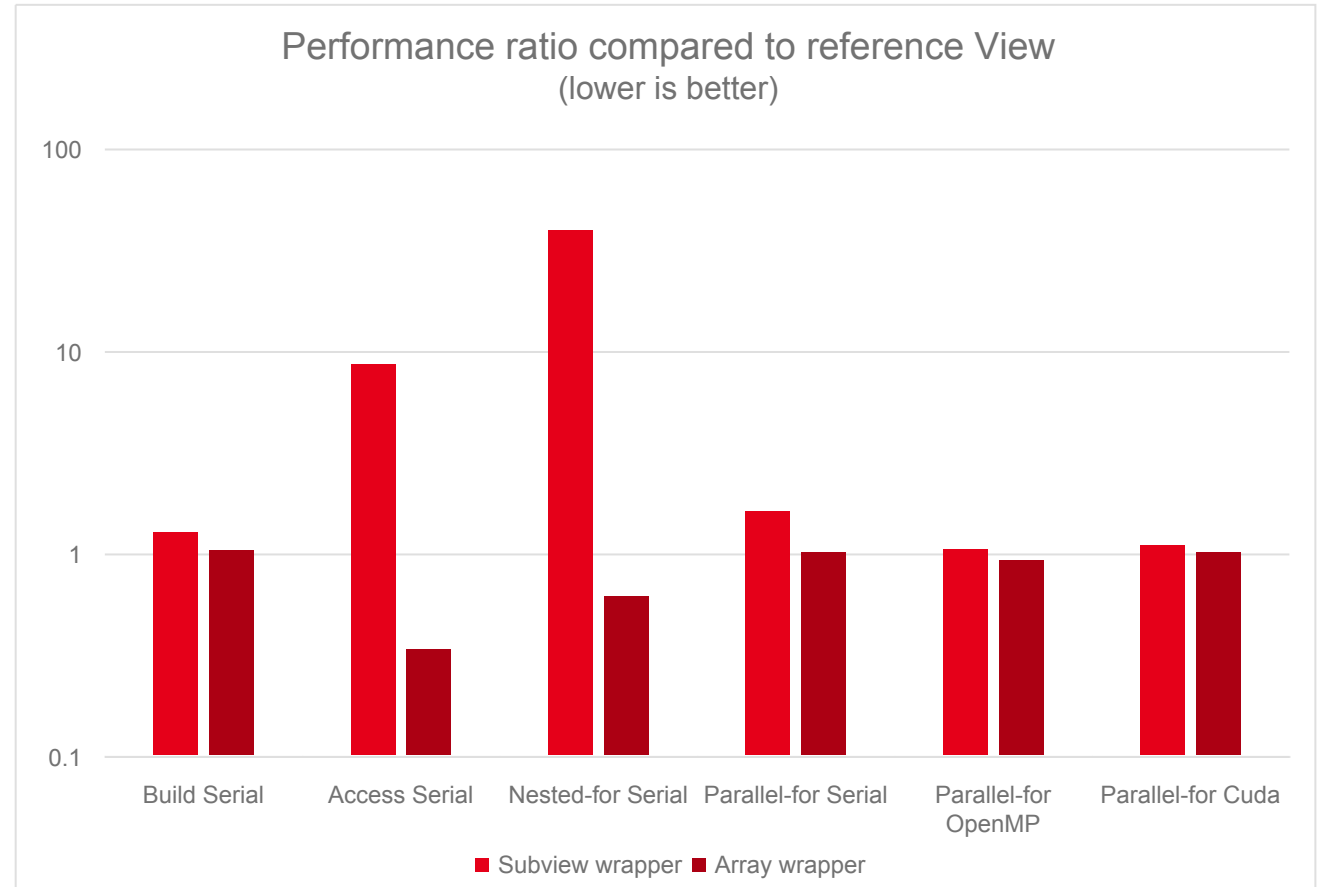
```
1. void myFunction(int *** data) {  
    data[i][j][k] = 10;  
}
```

```
2. void myFunction(Wrapper data) {  
    data[i][j][k] = 10;  
    data(i, j, k) = 10;  
}
```

```
3. void myFunction(View data) {  
    data(i, j, k) = 10;  
}
```

Development of a brackets wrapper library

- Brak helper
(**B**rackets operator for **K**okkos)
- Allows programmers to use Kokkos views like plain-old-data C arrays using either
 - A subview-based wrapper
 - An array-based wrapper (best performance)
- Overhead very acceptable



Conclusion

Provide support to migrate codes

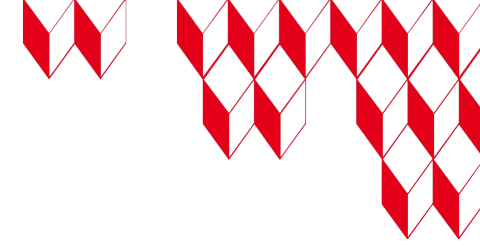
- Dynk helper (Dynamic Kokkos)
<https://github.com/cexa-project/dynk>
- Brak helper (Brackets operator for Kokkos)
<https://github.com/cexa-project/brak>



7 ■ Kokkos-Comm

Cédric Chevalier

Extending Kokkos to distributed computing



No primitives for distributed computing

HPC applications based on Kokkos must rely on external frameworks for distributed computing

- Generally message-passing based: MPI, NCCL, RoCCL, etc.
- Also an official remote memory access library: Kokkos-Remote-Spaces

Critical feature for most of our users

- GyselaX, Triclade, Trust, Thor, ...

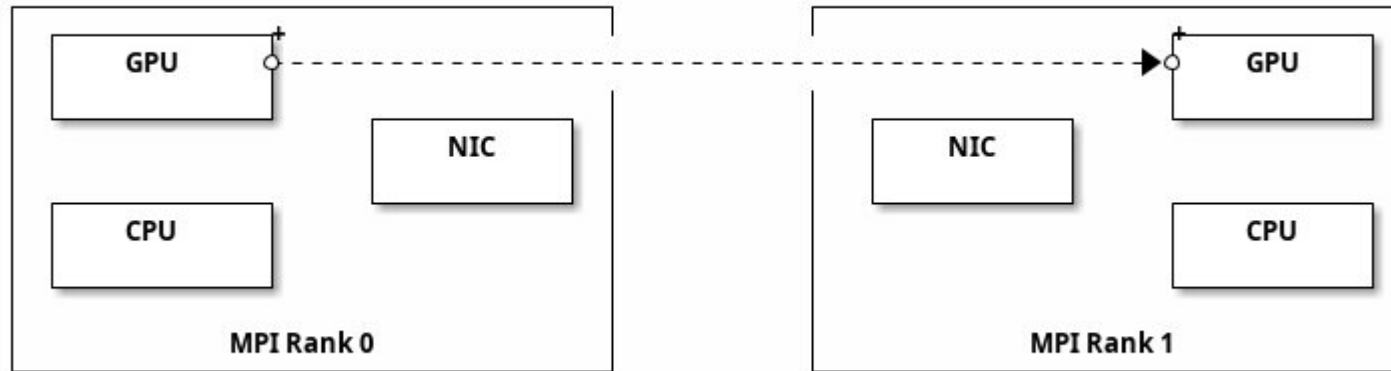
Challenges for programmers

- Must handle implementation-defined specificities
- Must handle non-contiguous Views
- Currently, partial support is scattered among user code

⇒ Lots of code duplication across Kokkos projects

Motivations

How do we communicate Kokkos::View between two MPI processes ?



- Must handle implementation-defined specificness
 - Is the MPI GPU-aware?
 - Explicitly copy Views from CPU \Leftrightarrow GPU?
- Must handle non-contiguous Views
 - Send as multiple "small" contiguous chunks
 - Pack/unpack as one "big" contiguous chunk
 - Use custom MPI DataType

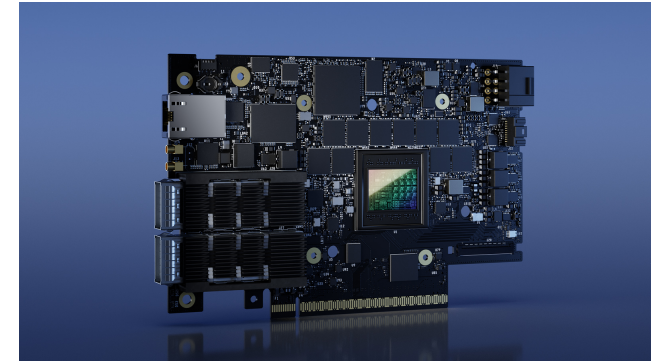
Solution: a common project, Kokkos-Comm

New project, with SNL, launched in March 2024

- Gather good MPI practice with Kokkos
- #mpi-interop on Kokkos's Slack
- Bi-weekly telecon, open to all (you are welcome to join!)

Be future proof

- Asynchronous communications (C++ 26)
- Optimized communications layers: NCCL, *CCL, RMA
- Optimized algorithms (MPC, BXI)
- Device-initiated communications
- Smart NIC
- MPI Forum for C++



Multi-Processor Computing



8 ■ Linear algebra and splines

Thomas Padioleau

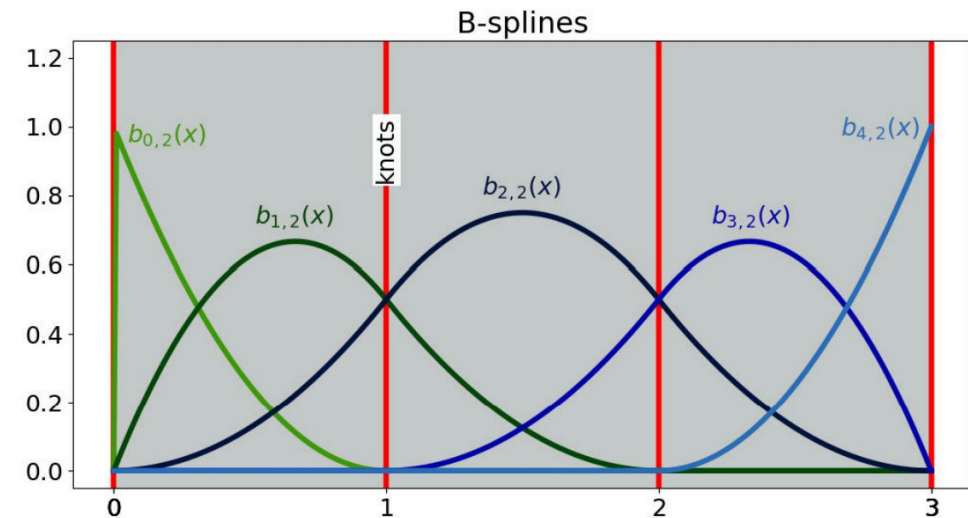
Gysela-X++, from the spline representation...

Multiple representations of the distribution function

- Point-wise representation
- Fourier representation
- Spline representation

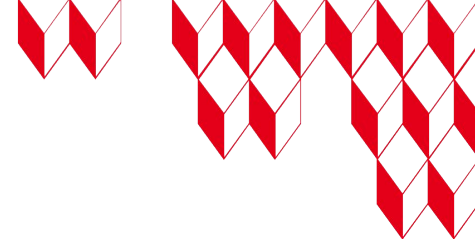
Splines

- smooth piece-wise polynomial functions



B-splines of degree 2 for the domain $[0,3]$
Credit: E. Bourne

... to linear systems



$$\begin{pmatrix} * & * & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & * \\ * & * & * & 0 & 0 & \cdots & 0 & 0 & 0 & * \\ 0 & * & * & * & 0 & \cdots & 0 & 0 & 0 & * \\ 0 & 0 & * & * & * & \cdots & 0 & 0 & 0 & * \\ 0 & 0 & 0 & * & * & \cdots & 0 & 0 & 0 & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & * & * & 0 & * \\ 0 & 0 & 0 & 0 & 0 & \cdots & * & * & * & * \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & * & * & * \\ * & * & * & * & * & \cdots & * & * & * & * \end{pmatrix} \begin{pmatrix} x_{11} & \cdots & x_{1n_b} \\ x_{21} & \cdots & x_{2n_b} \\ x_{31} & \cdots & x_{3n_b} \\ x_{41} & \cdots & x_{4n_b} \\ x_{51} & \cdots & x_{5n_b} \\ \vdots & & \vdots \\ x_{(n-3)1} & \cdots & x_{(n-3)n_b} \\ x_{(n-2)1} & \cdots & x_{(n-2)n_b} \\ x_{(n-1)1} & \cdots & x_{(n-1)n_b} \\ x_{n1} & \cdots & x_{nn_b} \end{pmatrix} = \begin{pmatrix} b_{11} & \cdots & b_{1n_b} \\ b_{21} & \cdots & b_{2n_b} \\ b_{31} & \cdots & b_{3n_b} \\ b_{41} & \cdots & b_{4n_b} \\ b_{51} & \cdots & b_{5n_b} \\ \vdots & & \vdots \\ b_{(n-3)1} & \cdots & b_{(n-3)n_b} \\ b_{(n-2)1} & \cdots & b_{(n-2)n_b} \\ b_{(n-1)1} & \cdots & b_{(n-1)n_b} \\ b_{n1} & \cdots & b_{nn_b} \end{pmatrix}$$

Matrix A

- medium size, $n \sim 10^3$
- almost banded

Right-hand side B

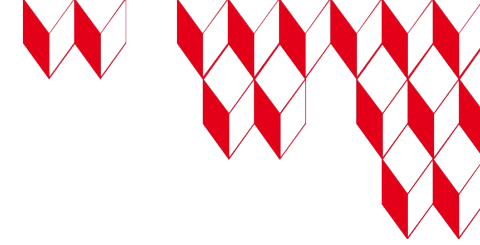
- Large number, $n_b \sim 10^7$
- Embarrassingly parallel in the batch direction

$$AX = B,$$

$$A \in M_n(\mathbf{R}),$$

$$X, B \in M_{n,n_b}(\mathbf{R})$$

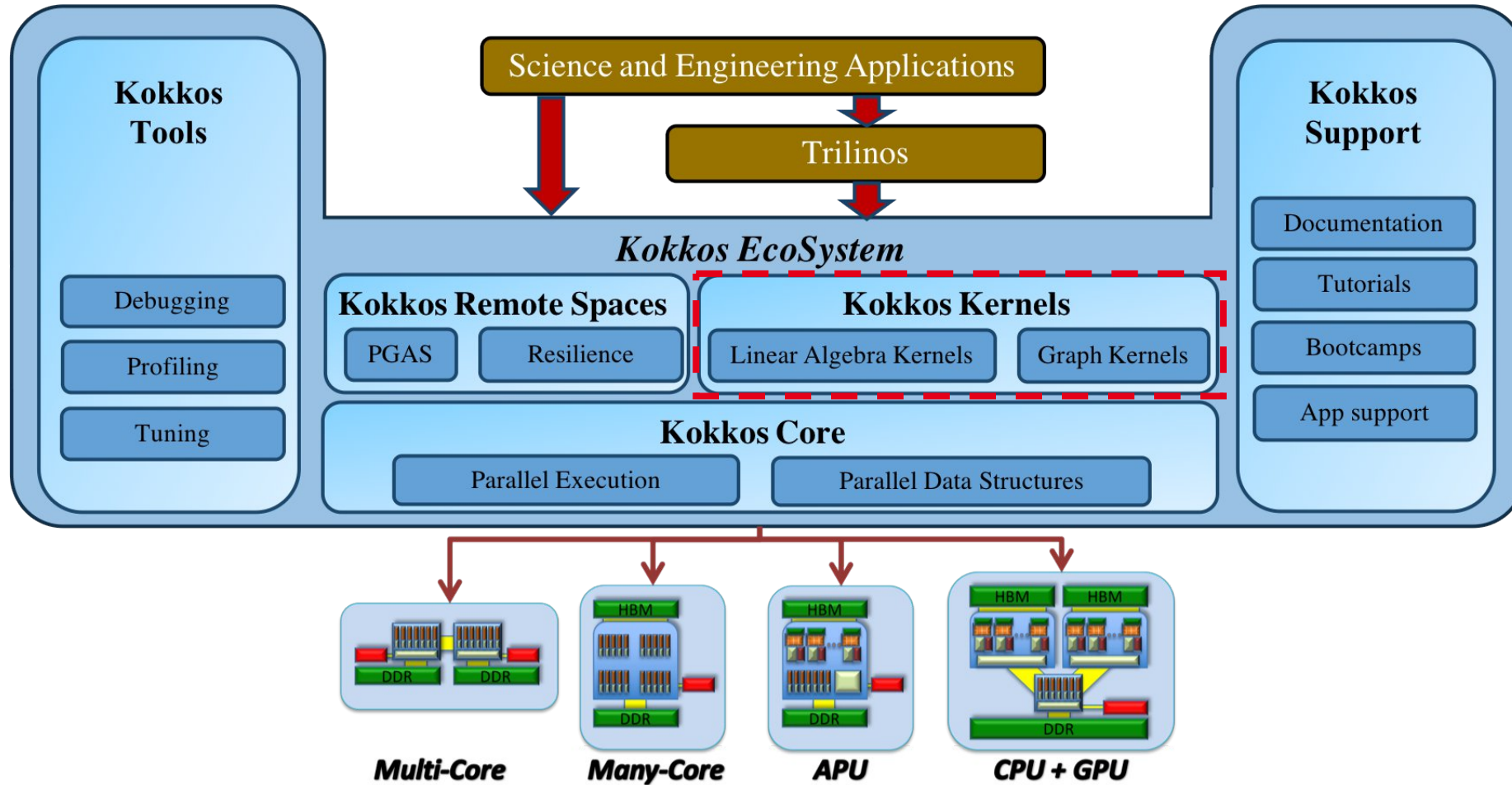
What solvers, what libraries ?



	Iterative method (Ginkgo)	Direct method
Pros	<ul style="list-style-type: none">• Works for any matrix• Existing efficient and parallel libraries developed by domain experts	<ul style="list-style-type: none">• Take advantage of the structure of the matrix both in terms of storage and computation• Fixed number of operations• For a given matrix, the factorization can be reused
Cons	<ul style="list-style-type: none">• Number of iterations can be large if the condition number is large• Does not benefit from the structure of the matrix	<ul style="list-style-type: none">• Difficult to find parallelism for one linear system• Multiple solvers to maintain

Experiments with Ginkgo **not satisfying**: must develop **new** solvers

The Kokkos ecosystem



CExA contributions to Kokkos-Kernels



- Serial device callable solvers any CPU/GPU
 - getrs: general matrix
 - gbtrs: general banded matrix
 - pbtrs: positive definite symmetric banded matrix
 - pptrs: positive definite symmetric tri-diagonal matrix
- Used in the library DDC on which Gysela-X++ is based
- These solvers can be reused by other applications
 - Small to medium size matrices
 - Large number of right-hand sides
- Ongoing work on developing factorization functions to ease deployment



9 ■ Conclusion

Julien Bigot

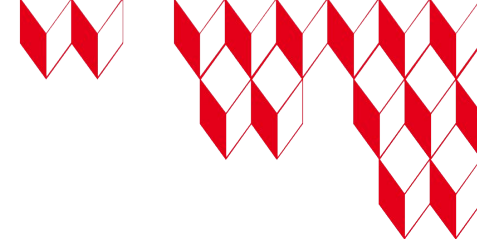
Feedback after one year

- A great support by the **Mission Numérique**
 - **Kick-off** funding with limited administrative overheads
- A **real need** in the community
 - For a **choice** to be made
 - Trainings are in **high-demand**
 - **New contacts** every month in **CEA**, but also **beyond**
 - In France: **CERFACS**, **CNRS**, **EDF**, **Inria**, **OCA**, **ONERA**, **Safran**, ...
 - Elsewhere: **BSC** (SP), **DoE** (US), **FZJ** (DE), **Riken** (JP), ...
 - Opened the door of **HPSF as a founding member**
- An extremely efficient way to **mutualize developments** between independent applications
 - Pair programming between CExA & app developers
 - **Shared features** end up in **shared libraries**, **support is done once only**
- Scaling & long-term funding of the project are our next challenges

CExA: Next steps

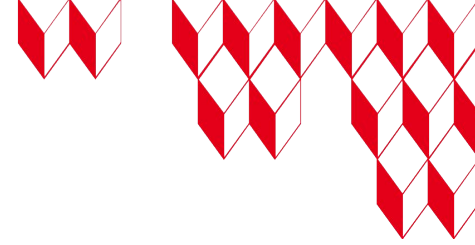
- **Q4 2024:** Full Kokkos-based release of GYSELA
- **Dec. 2024:** Select three more demonstrators
 - Submission of joint projects at PTC-SN
- **Q1 2025:** **Extend the project beyond CEA**
- **Q1 2025:** Full Kokkos-based release of Trust/TrioCFD
- **Q2 2025:** Kokkos-based run of Triclade
- **2025:** Selection of Alice Recoque architecture
 - **Focus optimizations** of the selected architecture
- **Q4 2025:** **Release** libraries optimized for Alice Recoque
- **2026:** **Large-scale execution** of our demonstrators on Alice Recoque

How to work with us



- **Fill our survey** and let us know about you
 - We will contact those who answered in early 2025
- **Join the community**
 - Get support on **slack**, join the French community on #general-fr
 - Register to the **mailing-list** to be notified of events
 - Attend our **monthly animations**
 - Virtual coffee 1st Monday of the month, 1PM (Paris time)
 - Virtual tea-time 3rd Wednesday of the month, 4PM (Paris time)
- Participate in our **trainings**
- **Submit a project** together
 - At **CEA**, **PTCs** in November
- Contact us and take part in the development of the libraries

The core team



Julien Bigot

Principal investigator



Ansar Calloo

Group leader



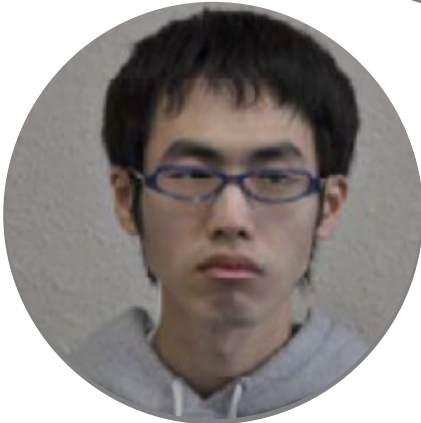
Cedric Chevalier

Group leader



Mathieu Lobet

Group leader



Yuuichi Asahi

Senior developer



Rémi Baron

Senior developer



Thomas Padioleau

Senior developer



Paul Zehner

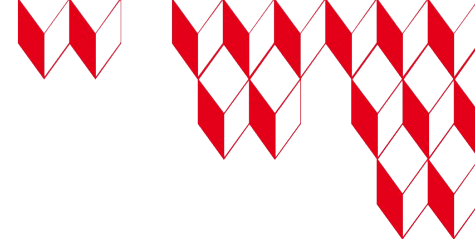
Developer



Hariprasad Kannan

Developer

The extended team



Julien Thelot

Administrative & animation support



Pierre Ledac

Trust/TrioCFD lead



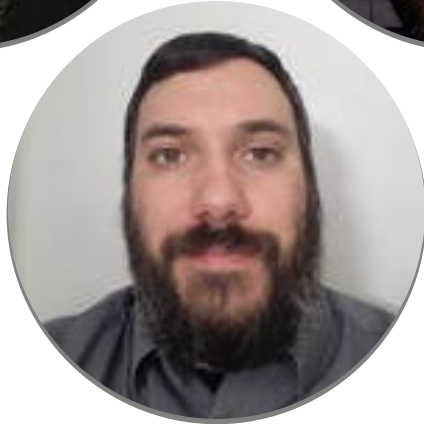
Virginie Grandgirard

GyselaX++ lead



François Letierce

Triclade lead



Julien Jaeger

DAM link



Édouard Audit

Network animator



Samuel Kokh

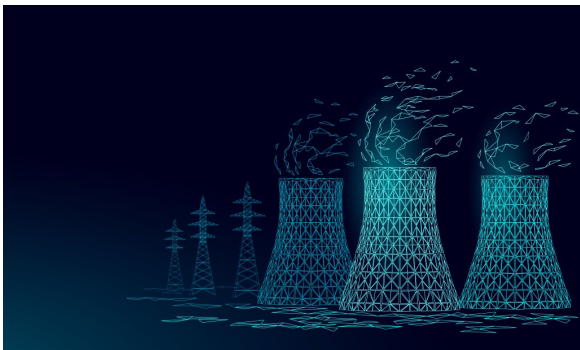
DES link



Patrick Carribault

DAM link

To conclude



- **Kokkos** is a strong **vendor-neutral**, **performance portable** Exascale programming model with **GPU** support
- 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**
- A **knock-on** effect with new **synergies** identified every month with code developers