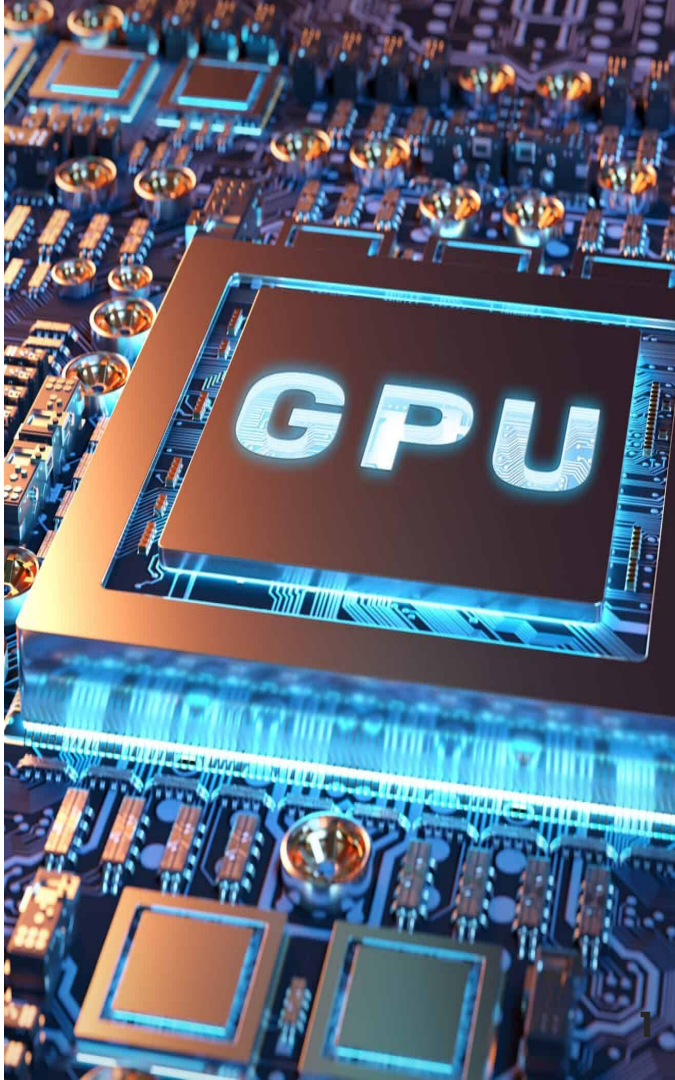
Kokkos, a library for performance portable GPU programming

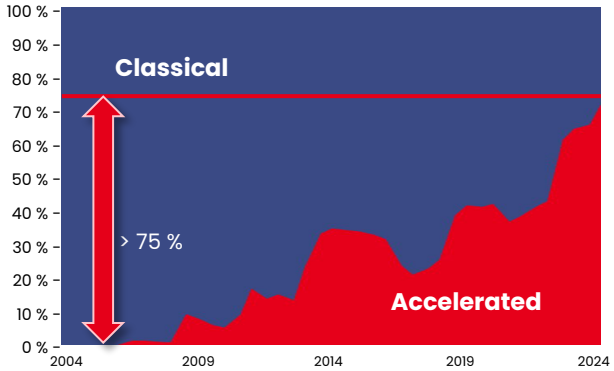
co-developed by CExA, a project for Computing at Exascale on Accelerators at CEA



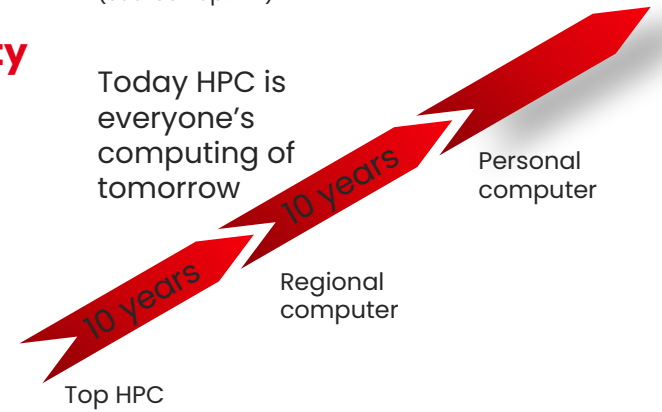
Seminar @ CELIA
September 25th 2024
Julien Bigot & CExA team

Context

- HPC is a tool in a wide range of domains, source of competitivity
  - 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)
  - 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
  - Trust in the roadmap



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



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

Performance

Performance portability

Domain abstractions

GPU transparency

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

Performance

Performance portability

Domain abstractions

GPU transparency

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

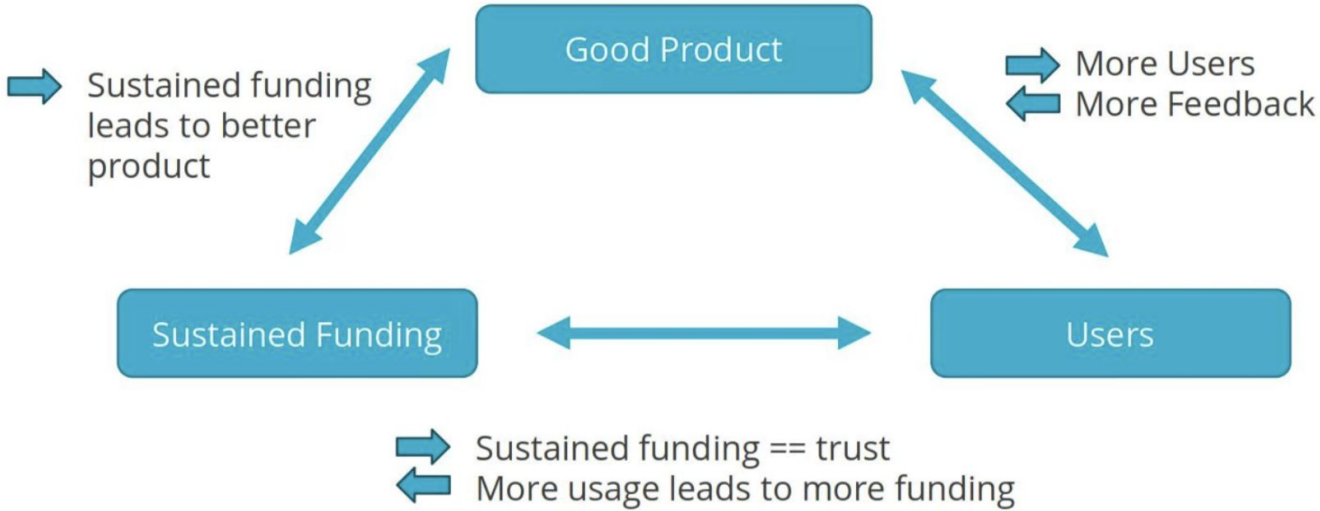
- Production grade, with public support
- Vendor neutral

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

With CExA, CEA chooses Kokkos



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

© Christian Trott & Damien Lebrun Grandie

CEx

eminate
d offer
ning for
opeans

Launched in september 2023, Publicly announced last ISC

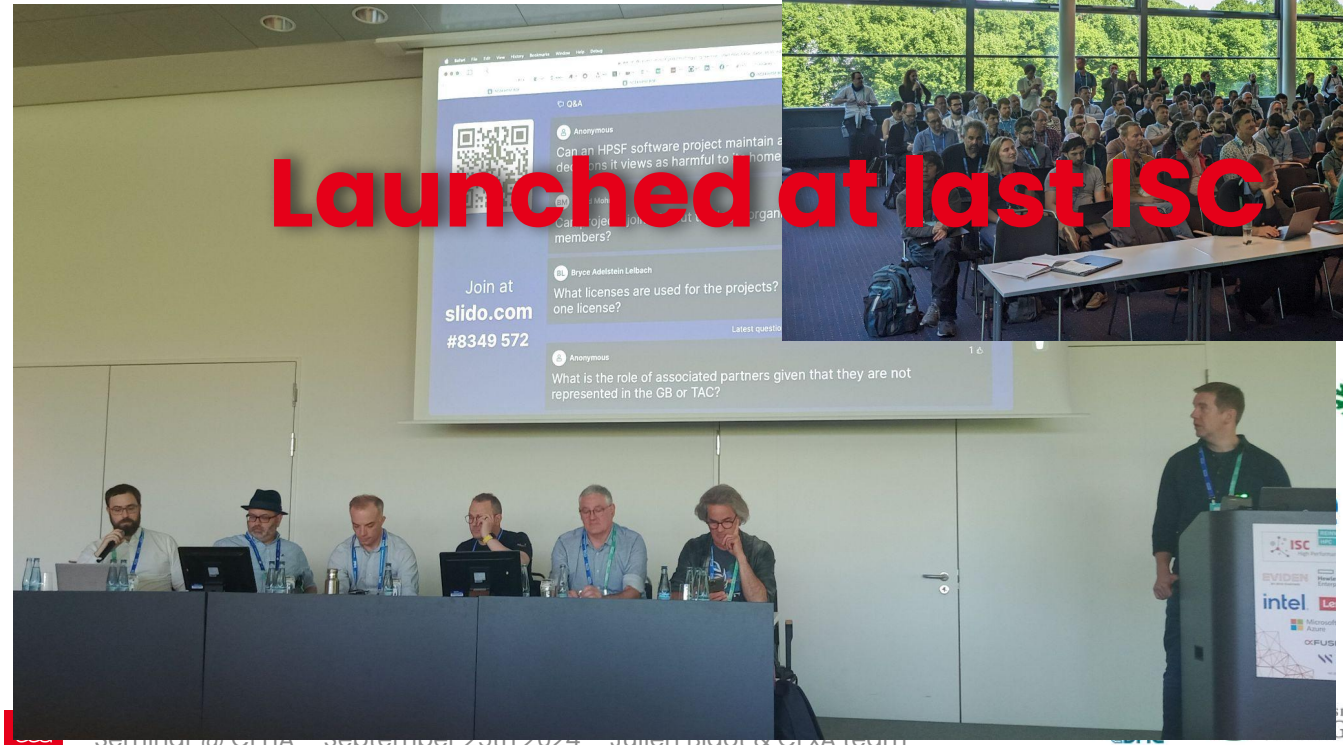
CExA in short

"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

HIGH PERFORMANCE SOFTWARE FOUN Members

HPSF Goals



Launched at last ISC

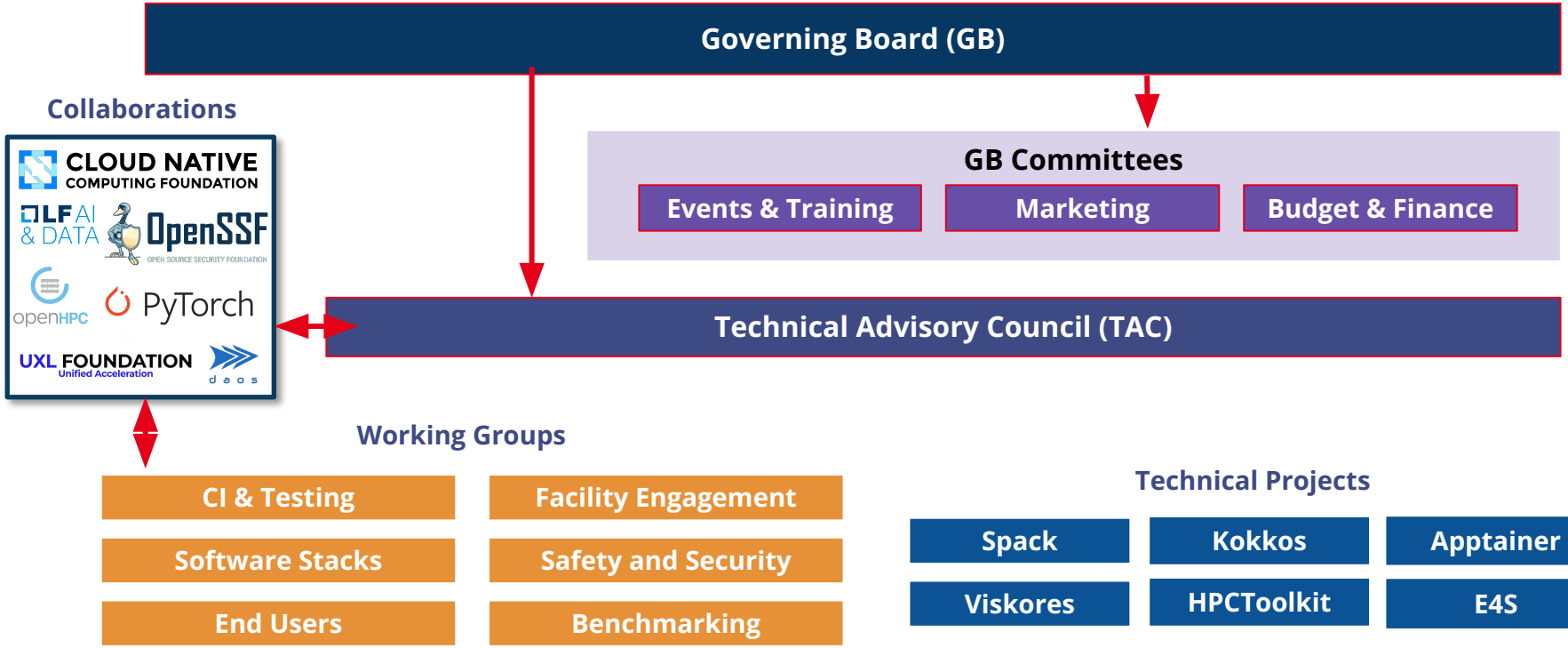
OAK RIDGE National Laboratory

Los Alamos NATIONAL LABORATORY

cea

UNIVERSITY OF MARYLAND

HPSF: the structure

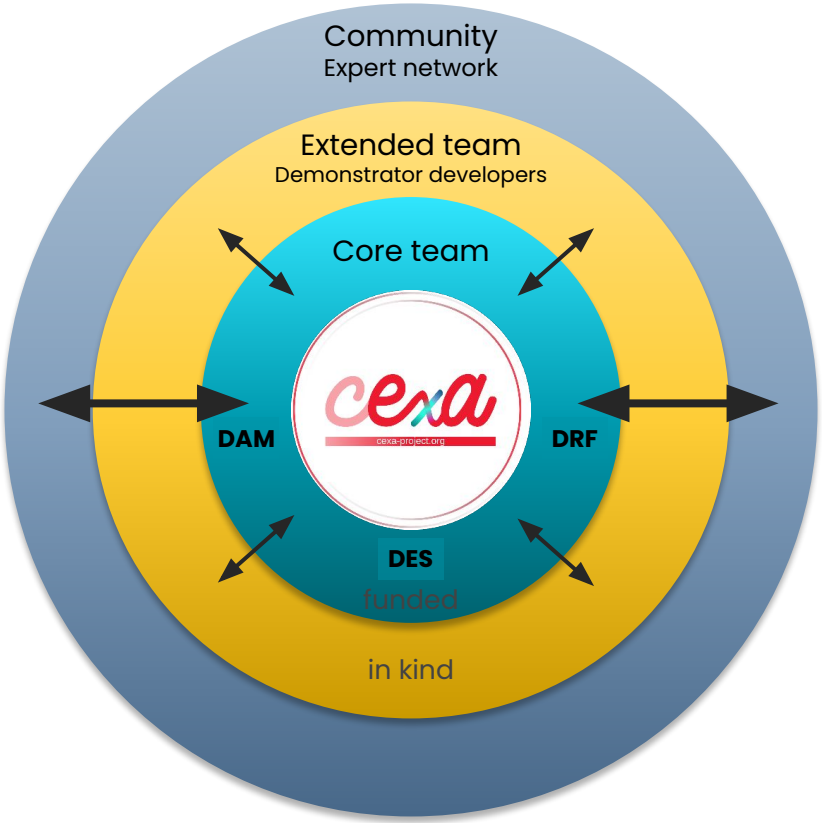


Two (independant) ways to participate

- Joining as a member
  - You need to join the Linux Foundation (Non-profit/academic, as associate for $0)
  - Joining HPSF at one of three levels:
    - Premier: $175k per year
    - General: $2.5k - $50k depending on size of organization
    - Associate: $0 for non-profit / academic
  - Take a stand, fund it & get a say on where the funding goes to

- Joining as a project
  - For the High Performance Computing ecosystem
  - That need a neutral home to facilitate multi-institutional collaborations
  - Providing vendor neutral solutions to engineering and science computational needs
  - Committed to building an open developer and user community

CExA project in practice



- **Core team**
  - Management, implementation and dissemination
  - 8 permanent researchers from all over CEA
  - 3 recrutements 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 C**ExA**:
    - Identification of needs
    - Usage of C**ExA** in applications
  - Priority target for dissemination
  - Sustainability of the work

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 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

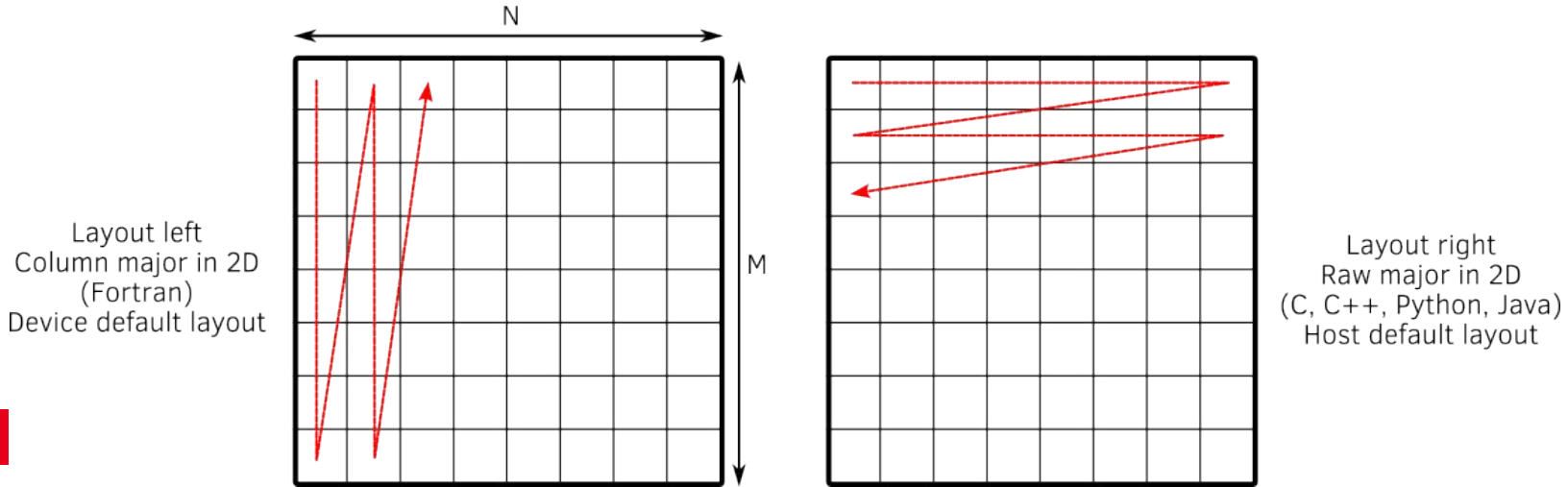
```
auto dview = create_mirror(mspace, a_view); // allocates & copy a new view of same size
auto dview = create_mirror_view_and_copy(mspace, a_view); // allocates & copy if necessary
```

- Allocates & copy to a new memory space

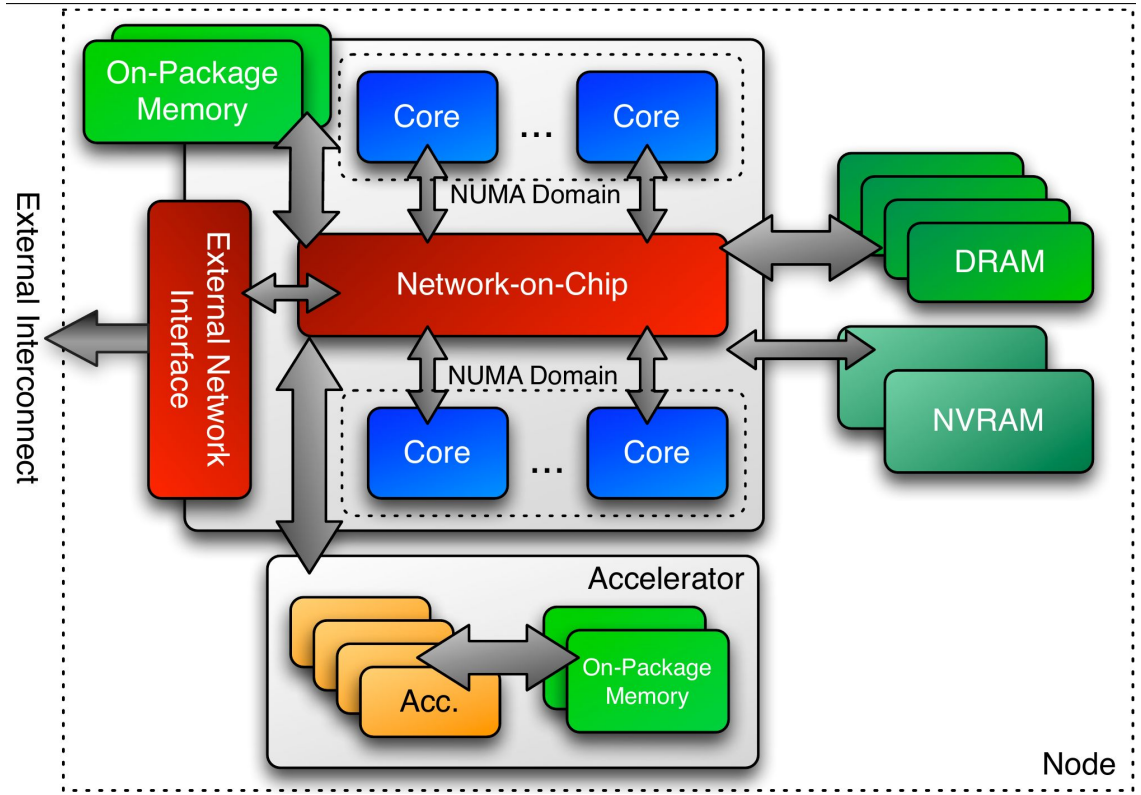
Kokkos views layout

```
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)



Kokkos hardware abstraction



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

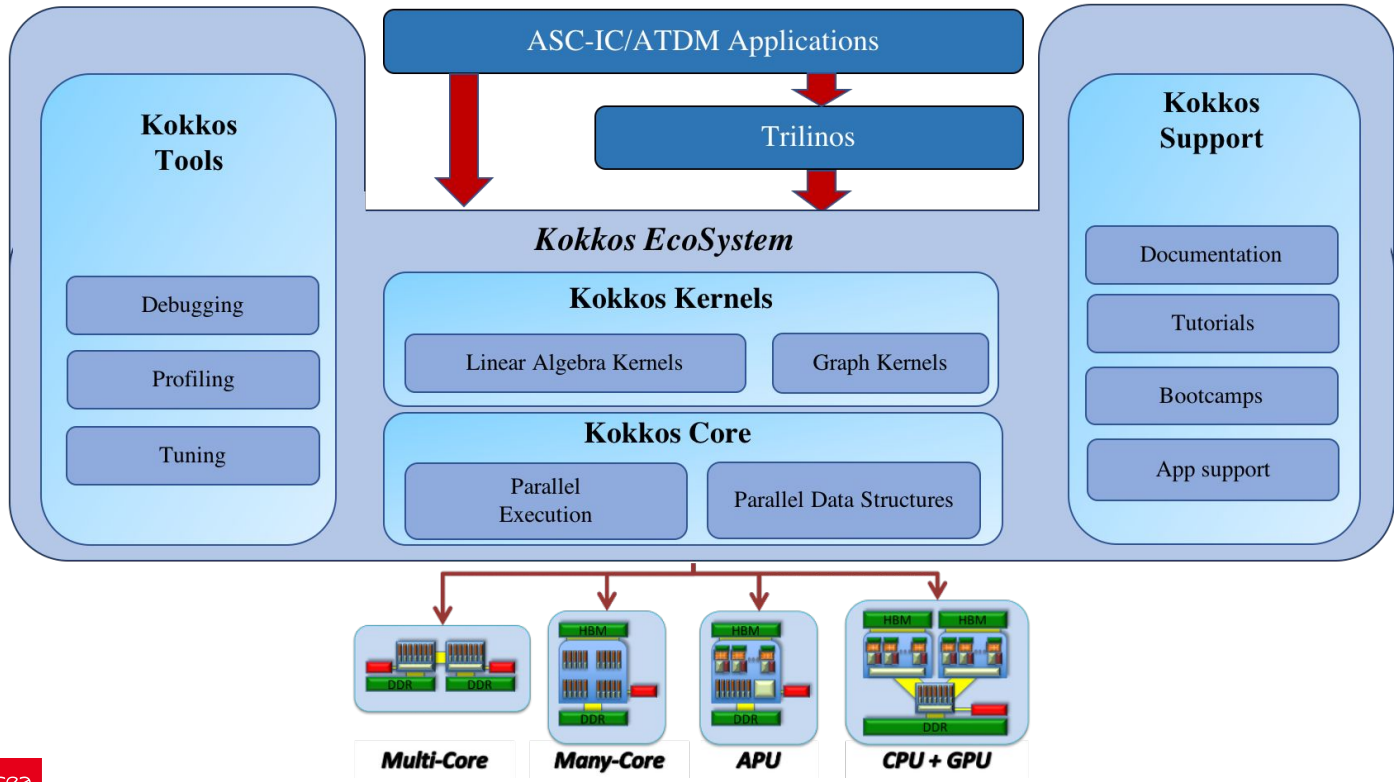
- Up-to-date tracking & automatic transfers when required

Scratch memory

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

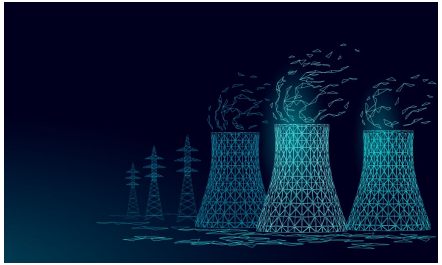
- **Parallel patterns w. asynchronous support**
  - **Independent interactions, Reductions, Scans**
- Iteration strategies
  - Tiled, Hierarchical, …
- Algorithms
  - Sorting
  - Random number generation
  - Most of STL parallel algorithms
  - …
- QoL features: portable printf, etc.
- Portable atomic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

Kokkos Ecosystem



- Kokkos-FFT
- Kokkos-Comm
- Kokkos-Resilience
- ...

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 weeks with code developers

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 handling
- Add our contributions to Kokkos ecosystem
  - DDC
    - Discrete data & computation
  - Kokkos-FFT
    - Performance portable FFT with a Kokkos API
  - Kokkos-comm
    - Find out more in programming model session

Taking part in Kokkos weekly developers meetings

Kokkos training & community animation

- First training with Christian Trott & Damien Lebrun last september in Saclay
  - >80 participants
- Kokkos slack now has a `#general-fr` channel (~10% of the whole community)
- CExA virtual café once a month
  - Informal presentations & discussions, in French about Kokkos, its ecosystem & GPU at large

- Kokkos virtual tea-time once a month
  - Informal presentations & discussions, in English about Kokkos, its ecosystem & GPU at large
  - With our US partners

- Next Kokkos training on 17-19 June @ Saclay
  - with Damien Lebrun & Luc Berger-Vergiat
  - Registrations still open
- CEA / EDF / Inria summer school in summer 2025

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

Thomas Padioleau
Developer

The extended team

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

Join us & join the fun!

2-years HPC DevOps Engineer position

Deployment and CI on supercomputers for the C++ Kokkos library within the “Moonshot” CExA project

CEA is recruiting DevOps engineers for a 2-year period to join the CExA “Moonshot” project team, which is setting up CEA’s GPU computing software stack around the Kokkos C++ library, to contribute to innovative packaging, deployment and continuous integration approaches for supercomputers, based in particular on Spack. A team of more than 10 people is currently being set up. The positions will be based at the CEA Saclay site near Paris.

2-years C++ expert engineer position

Contribution to the development of the Kokkos GPU computing library within the CExA “Moonshot” project

Join the CEA’s ambitious “Moonshot” project, CExA, and contribute to the development of the Kokkos GPU computing library. We are recruiting six talented and enthusiastic C++ development engineers for a period of 2 years to work at our CEA Saclay site near Paris.

https://cexa-project.org