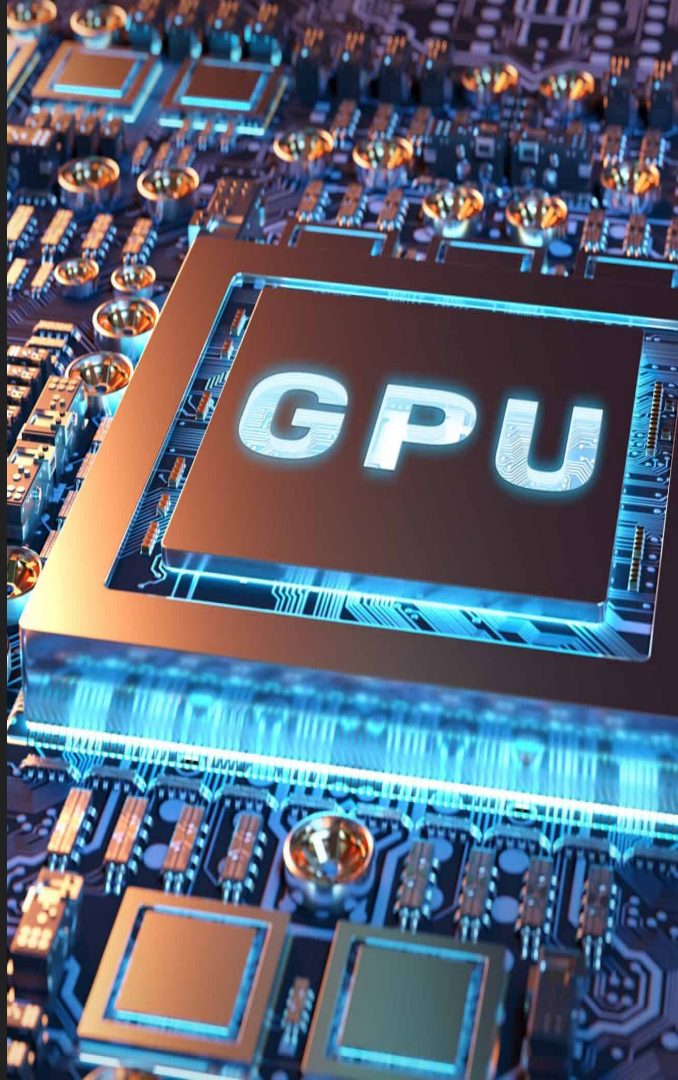
Programming GPUs at the parallel loop level: The case of Kokkos



NumPEx GPU Workshop
June 12th 2024
Julien Bigot & CExA team

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
- 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, …

Performance

Performance portability

Domain abstractions

GPU transparency

Available solutions

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

Available solutions

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

- **Production grade, with public support**

Available solutions

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

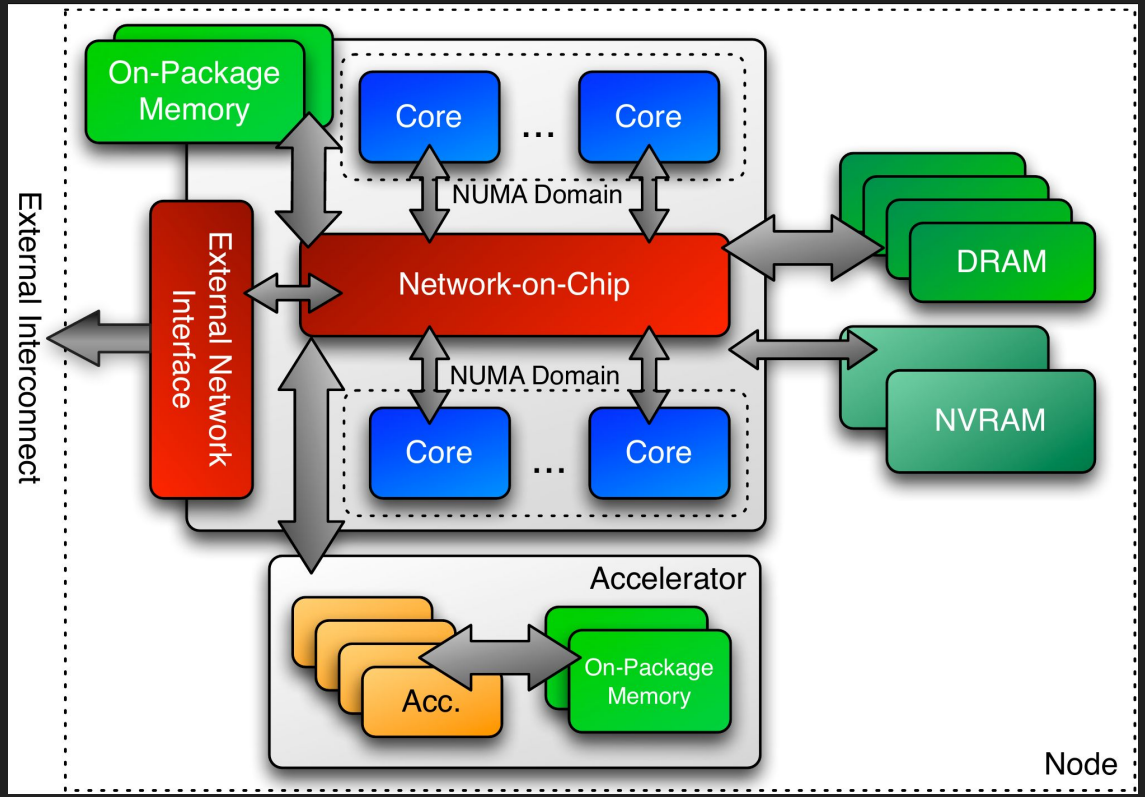
- Production grade, with public support
- **Vendor neutral**

Available solutions

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

- Production grade, with public support
- Vendor neutral

Kokkos hardware abstraction



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

```
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

```
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

- 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

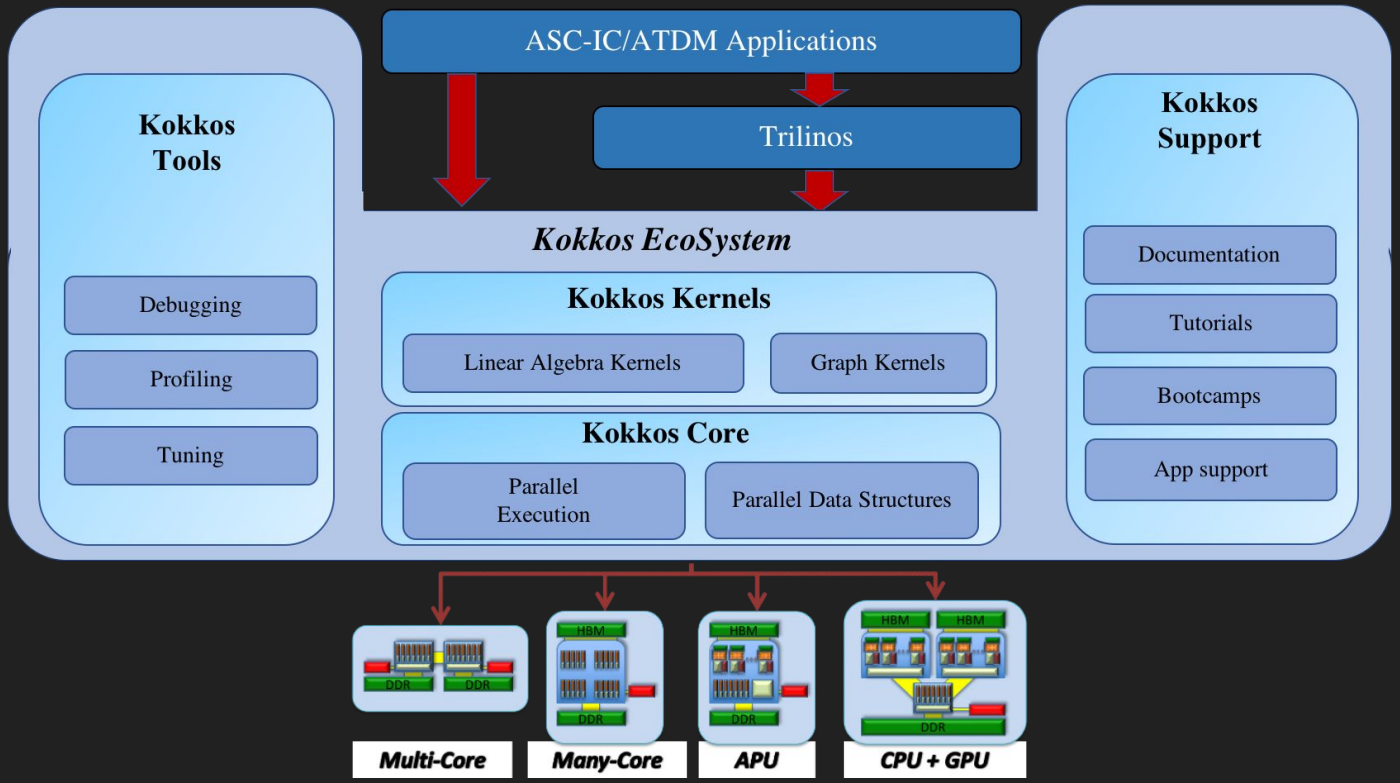
- A C++ template library
- No direct code generation, rely on vendors C++-like languages
- Multiple “backends”, selection at compile time
  - OpenMP, Cuda, OneAPI, HIP, …
- Maximum 3 backends enabled at once
  - Serial backend
  - 1 Host parallel backend
  - 1 Device parallel backend

Also 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
+ …

Available solutions

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

- Production grade, with public support
- Vendor neutral

Available solutions

- Cuda
- HIP/ROCm
- **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, the CEA chooses Kokkos

To help applications move to GPU at CEA, in France, and in Europe

"**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: a 15 people team to work on Kokkos and its ecosystem

Kokkos and the C++ standard

A window in the future of Parallel C++

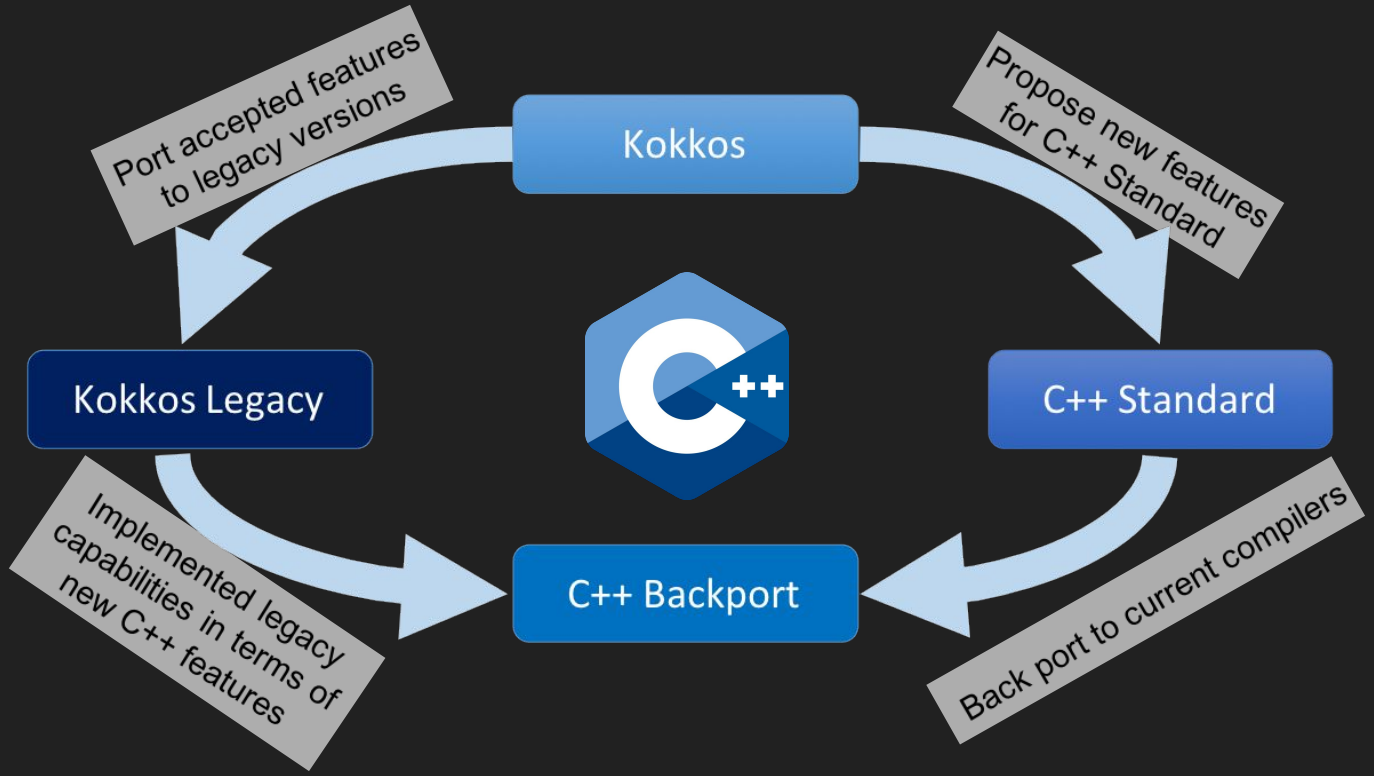
Done:

- Mdspan
- Lin alg
- Atomics

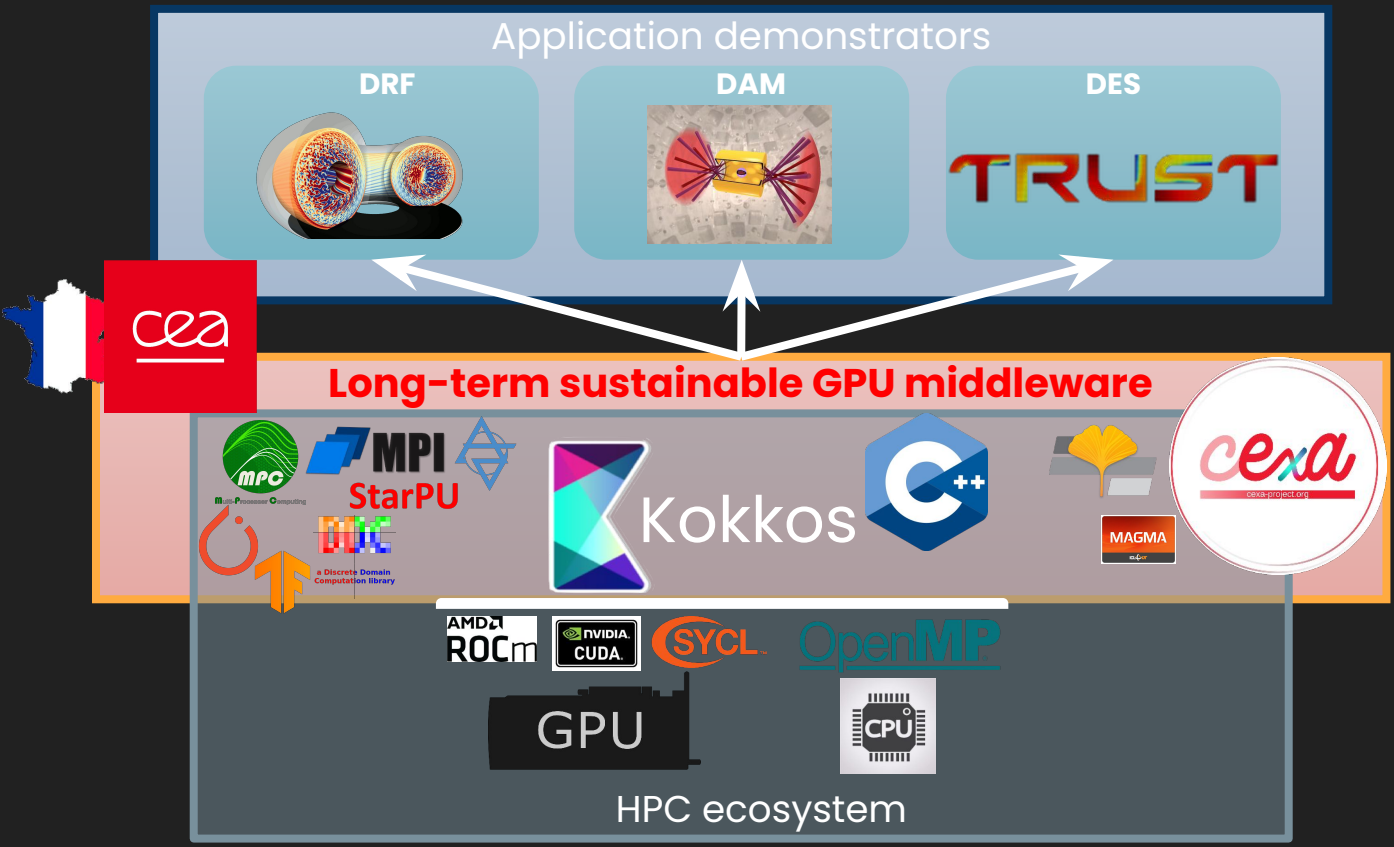
Next:

- SIMD
- …

In a backward compatible way



The strategy of CExA



Disseminate and offer training to the community

Adapt application demonstrators

Contribute to a long-term sustainable software middleware for GPU computing

Join us & join the fun!

https://cexa-project.org

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.

Next Kokkos training on 17-19 June @ Saclay with Damien Lebrun & Luc Berger-Vergiat

https://indico.math.cnrs.fr/e/kokkos_days