DDC, a performance portable library abstracting Computation on Discrete Domains

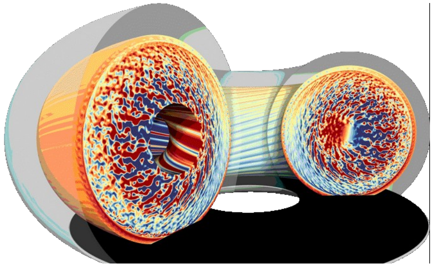
Thomas Padioleau[1], Julien Bigot[1], Baptiste Legouix[2]

1. Maison de la Simulation, CEA Paris-Saclay

2. IRFM, CEA Cadarache

This work was supported by ANR PIA funding: ANR-20-IDEES-0002

What is GYSELA ? – Physics



- Plasma physics for nuclear fusion
- Study core turbulence in tokamaks
- Distribution function f_s(x,v,t) for each species s
- Solves Vlasov-Poisson equations (7D) in the gyrokinetic approximation (6D)

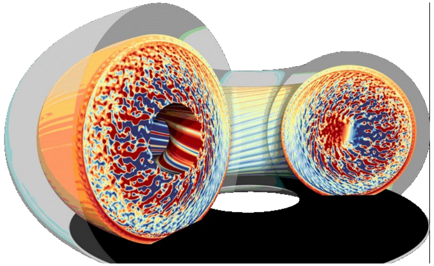
What is GYSELA ? – Software

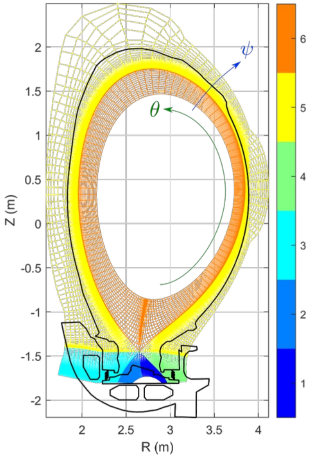
- ≈25 years old Fortran 90 code (≈ 40 kLOC) developed by a team at CEA/DRF/IRFM
- Distributed memory parallelism : MPI
- Shared memory parallelism : OpenMP
- Fine tuned for (Intel) CPUs
- Efficient simulations on 100K cores, 100M CPU hours

Goals

Rework GYSELA to:

- Prepare the Exascale
  - Multi vendor GPU (attempt with OpenMP target)
  - ARM A64FX
- Handle complex geometry: realistic tokamak
- Handle multiple (and different)
  - Discretizations
  - Domain decompositions



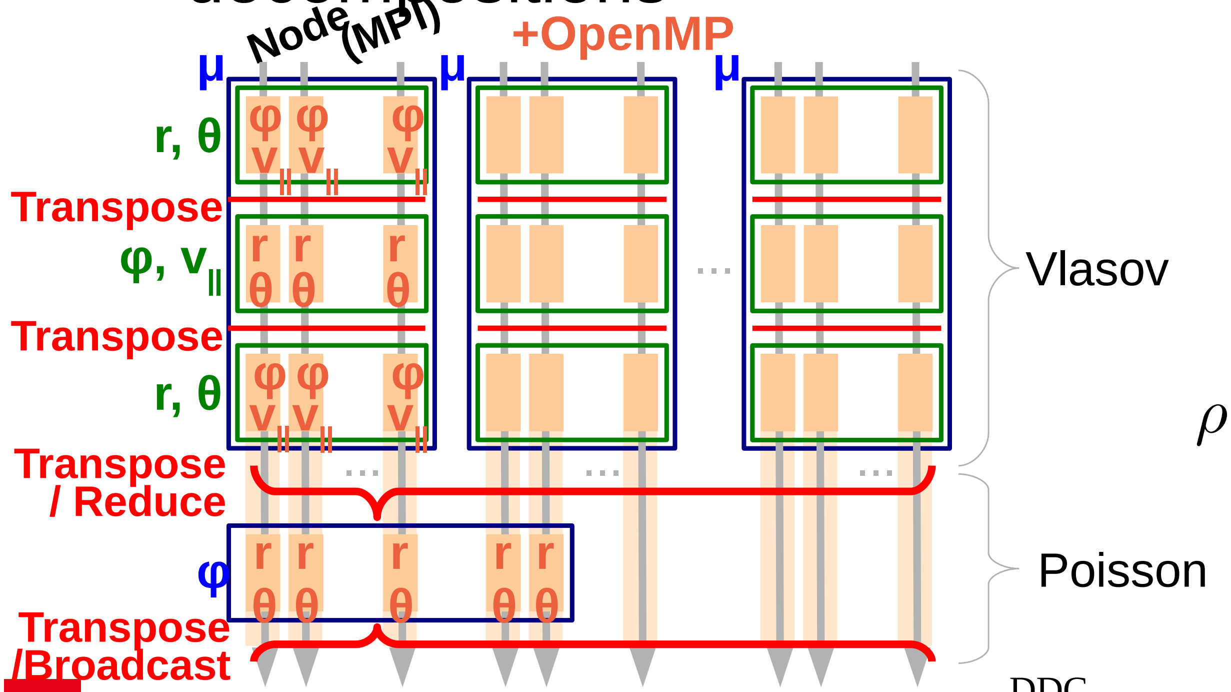


Implicits – Arrays

- Multiple representations of the distribution function
  - Spline representation
  - Fourier representation
  - Pointwise representation
- Multiple meshes
  - Uniform
  - Non-uniform

Implicits – Evolving domain decomposition

- Multiple MPI domain decompositions



- Multiple dimensionnalities
  - Distribution function

$$f_s(r, \theta, \phi, v_\parallel, \mu, t)$$

  - Charge density

$$\rho(r, \theta, \phi, t) = \int q_s f_s(r, \theta, \phi, .., .., t)$$

Implicits

- An index is… an int, no context:
  - From what discretization ?
  - From what decomposition ?
  - With a halo zone ?
  - ...
- Information about these using global variables

→ ~~Impossible~~ Difficult to change the code
Discrete abstraction : DDC

What is DDC ?

- Started in May 2021 with *Julien Bigot*
- Inspired by the python library xarray that uses labeled dimensions
- C++17 library supporting « zero-overhead » dimension labeling for multi-dimensional arrays and performance-portable multi-dimensional algorithms
- Based on Kokkos

a Discrete Domain Computation library

Core concepts: ddc::DiscreteDomain

- For each new (discrete) dimension : create a tag
  - class DDim0, class DDim1, ...
- A (discrete) dimension is identified by a type
  - DiscreteDomain<DDim0>, DiscreteDomain<DDim1>, …
- Multi-dimensional domain:
  - DiscreteDomain<DDim0, DDim1>

Core concepts: ddc::DiscreteElement

- An instance of this type is a multi-dimensional range, an interval of DiscreteElement
  - DiscreteDomain<DDim0> dim0_dom = … ;
  - for (DiscreteElement<DDim0> const& dim0_elem : dim0_dom) { … }
- Difference between two DiscreteElement is a DiscreteVector
- Possiblity to associate static constant attributes to DiscreteElement or a whole dimension
  - DiscreteElement<DDim0> **dim0_elem**;
  - get_constant_data_from(**dim0_elem**);
  - get_other_constant_from<**DiscreteDomain<DDim0>**>();

Containers: ddc::Chunk, ddc::ChunkSpan

- Associates DiscreteElement to a value
  - Chunk<double, **DiscreteDomain<DDim0, DDim1>**> d01_chk(“label”, **dim01_dom**);
  - d01_chk(dim01_elem) = 99.9;
- Slicing feature
  - ChunkSpan<double, DiscreteDomain<**DDim1**>> d1_chk = d01_chk[dim0_**elem**];
  - ChunkSpan<double, DiscreteDomain<**DDim0, DDim1**>> d01_chk2 = d01_chk[dim0_**dom**];
- Similar to std::mdspan and Kokkos::View
- Let **dim0_elem** be a DisreteElement<DDim0>, **dim1_elem** a DiscreteElement<DDim1>, **dim01_elem** a DiscreteElement<DDim0, DDim1>
  - d01_chk(**dim0_elem**, **dim1_elem**) ✓
  - d01_chk(**dim1_elem**, **dim0_elem**) ✓
  - d01_chk(**dim01_elem**) ✓
  - d01_chk(**dim0_elem**, **dim0_elem**) ×, detected at compile-time

Core concepts: discretizations

- Discretization: generator of discrete dimensions
- DDC provides some discretizations
  - UniformPointSampling/ NonUniformPointSampling
  - PeriodicSampling
  - UniformBSplines/NonUniformBSplines
- Users can provide their own discretizations

- Create a tag for the continuous dim
  - class CDim0;
- Define the Discrete dimension tag based on this
  - using DDim0 = UniformPointSampling<CDim0>;
  - DiscreteDomain<DDim0>
- Use discretization attributes:
  - DiscreteElement<DDim0> dim0_elem;
  - coordinate(dim0_elem) → double;

Multi-dimensional algorithms

- Iteration over DiscreteElement:
  - **for_each**, similar to **Kokkos::parallel_for**
  - **transform_reduce**, similar to **Kokkos::parallel_reduce**
  - **fill** and **deepcopy**, similar to **Kokkos::deep_copy**
- Conversion of discrete dimensions:
  - ChunkSpan<T, InputDiscreteDomain> → ChunkSpan<U, OutputDiscreteDomain>
  - Discrete Fourier transform (FFT)
  - Spline transform (linear system)
- Performance portability is achieved through different libraries :
  - Kokkos, Ginkgo, cufft, hipfft, fftw

What DDC is *not* ?

- No math nor physics related operators :
  - No advection equation solver
  - No Poisson equation solver
- Not a mesh generation/partitioning library

Heat equation example – Discretization

```cpp
struct X; struct Y; struct T;

using DDimX = UniformPointSampling<X>; using DDimY = UniformPointSampling<Y>; using DDimT = UniformPointSampling<T>;

auto [x_domain, ghosted_x_domain, x_pre_ghost, x_post_ghost]
    = init_discrete_space(DDimX::init_ghosted(Coordinate<X>(x_start), Coordinate<X>(x_end), DiscreteVector<DDimX>(nb_x_points), gwx));

auto [y_domain, ghosted_y_domain, y_pre_ghost, y_post_ghost]
    = init_discrete_space(DDimY::init_ghosted(Coordinate<Y>(x_start), Coordinate<Y>(x_end), DiscreteVector<DDimY>(nb_y_points), gwy));

DiscreteDomain<DDimT> time_domain
    = init_discrete_space(DDimT::init(Coordinate<T>(start_time), Coordinate<T>(end_time), nb_time_steps + 1));
```

Heat equation example – Memory allocation

```
Chunk ghosted_last_temp(DiscreteDomain(ghosted_x_domain, ghosted_y_domain),
                        device_allocator<double>());


Chunk ghosted_next_temp(DiscreteDomain(ghosted_x_domain, ghosted_y_domain),
                        device_allocator<double>());
```

Taking advantage of the C++17 CTAD feature

Greatly simplifies the syntax

Heat equation example – Time loop

```
for (auto iter : time_domain) {
   std::cout << "Current time: " << coordinate(iter) << std::endl;
   deepcopy(ghosted_last_temp[x_pre_ghost][y_domain], ghosted_last_temp[y_domain][x_domain_end]);
   // other boundaries...

   ChunkSpan next_temp = ghosted_next_temp[x_domain][y_domain];
   double dx = step<DDimX>(), dy = step<DDimY>();
   double dt = distance_at_left(iter);
   for_each(policies::parallel_device, next_temp.domain(), KOKKOS_LAMBDA(DiscreteElement<DDimX, DDimY> ixy) {
               auto ix = select<DDimX>(ixy);
               auto iy = select<DDimY>(ixy);
               next_temp(ix, iy) = last_temp(ix, iy);
               next_temp(ix, iy) += (kx * dt / (dx * dx)) * (last_temp(ix + 1, iy) - 2.0 * last_temp(ix, iy) + last_temp(ix - 1, iy));
               next_temp(ix, iy) += (ky * dt / (dy * dy)) * (last_temp(ix, iy + 1) - 2.0 * last_temp(ix, iy) + last_temp(ix, iy - 1));
           });
}
```

« Simple » implementation with type safety from DDC

Conclusion

- Some of the implicits have been tackled by introducing compile-time labeled dimensions
- Future directions for DDC ?
  - Allow nested calls of algorithms
  - PGAS-like for MPI
    - DDC-based application would not call MPI
    - ask for a change of domain decomposition
  - Enrich the set of discretizations
  - Study performance/potential overheads

Thank you for your attention!

Github DDC: https://github.com/CExA-project/ddc

Slack : https://ddc-lib.slack.com

DDC website : https://ddc.mdls.fr

Github Gyselalibxx: https://github.com/gyselax/gyselalibxx