MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

Contributions to the GRILLIX and GEMPICX projects

Cristian-Constantin Lalescu
ACH-MPG (NMPP, IPP Garching)

EUROfusion

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Overview

GRILLIX

- tokamak/stellarator, various fluid models
- flux-coordinate-independent
- Fortran 2008, MPI
- depends on PARALLAX, also used by GENE-X
- CUDA/HIP acceleration with PAccX library

tasks

- port to GPU
- refactor/rewrite in C++/Kokkos

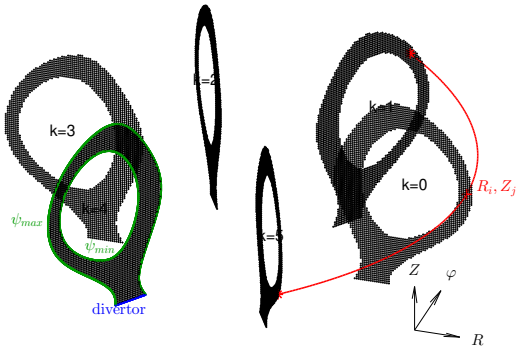
GEMPICX

- various models based on Vlasov equation
- geometric particle-in-cell
- multi-resolution through AMReX
- C++20, MPI
- GPU support through AMReX and Kokkos

tasks

- detailed performance analysis
- tweak/refactor code as needed
- extend IO/refactor for checkpointing

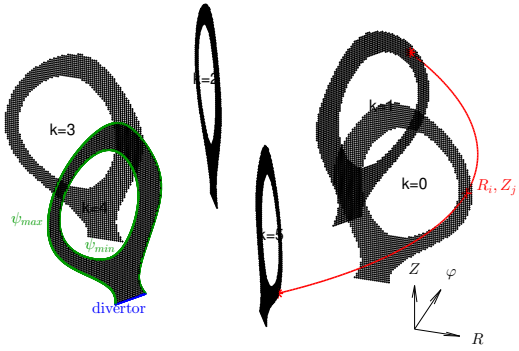
GRILLIX: PARALLAX and PAccX



Numerical framework

- geometry handled by PARALLAX:
  - small number of slices along toroidal direction
  - unstructured 2D grid
  - 1 MPI process per slice
- in-plane field equation accelerated with PAccX (PARALLAX Accelerator)
- perpendicular operators: in-plane computation
- parallel operators: MPI computation along red curve

PARALLAX: FCI



FCI

- 2D grid: subset of regular rectangular grid
- precomputed matrices store field line topology
- parallel operator
  - transfer data between planes
  - multiply with backward/forward matrices
  - perform stencil operation
- matrix preparation implementation in PARALLAX, used by GRILLIX and GENEX

GrillAccX — original plan

PAccX approach

1. init (f90 objects)
2. declare (C/C++ objects)
3. PAccX-init (C/C++ objects, f90 objects)
4. main loop
   - 4.1 compute-1 (f90 objects)
   - 4.2 PAccX-update (C/C++ objects, f90 objects)
   - 4.3 PAccX-compute(C/C++ objects, f90 objects)
   - 4.4 compute-3 (f90 objects)
5. PAccX-destroy(C/C++ objects)
6. finalize (f90 objects)

GrillAccX approach

1. init (f90 objects)
2. declare (C/C++ objects)
3. GrillAccX-init (C/C++ objects, f90 objects)
4. main loop
   - 4.1 GrillAccX-compute(C/C++ objects, f90 objects)
5. GrillAccX-destroy(C/C++ objects)
6. finalize (f90 objects)

Status: prototype CPU implementation passes integration tests.

PAccX/GrillAccX — new plan

Context

1. use case:
   - PAccX — "black box" with C API
   - GrillAccX — users should be able to add models
2. the presence of a GrillAccX C API requires additional complexity
3. new data structures required — additional code for host/device memory management

New plan: everything in C++ with Kokkos

1. `Kokkos::View` solves memory management
2. `Kokkos::parallel_for` typically enough for performance
3. single language — less complexity
4. pure CUDA extensions possible if necessary

PHOENIX/PAccX refactoring progress

Procedure

- one component at a time
- prototype CPU implementation passes tests
- refactor with Kokkos, minimal changes to tests

Status

- done: geometry/math
- done: interpolation
- on-going: equilibrium, field line tracing
- todo: mesh generation
- todo: refactor 2D solver
- todo: evaluate/redesign GrillAccX

Comments on Kokkos and C++20

1. dynamic linking leads to crashes:
   - switch to static
   - implement full functionality
   - fix cmake setup in future if necessary
2. virtual functions don't work in Kokkos
   - switch to C++20 concepts
   - use templates instead of "pointers to abstract base"

PHOENIX/PAccX notes

Spline interpolation

- PAccX is MIT licensed
- GSL is GPL licensed
- stand-alone implementation to compare backends
- stand-alone implementation agrees with GSL
- currently using minimal version in PAccX
- highlight: deduced analytic expression for linear system to solve for classic splines

TOML parameter files

- "Tom's Obvious Minimal Language"
- workaround for "C++ is missing namelists"
- found "toml11" on github, author is apparently biologist
- found "toml++" on github, author is random programmer
- test with arrays in input file:
  - toml11: approx 90s
  - toml++: approx 6s
  - netcdf: approx 1s
- wrote own faster parser, but it can't read matrices (yet).

But toml++ has 20000 LOC, can I trust it?

GEMPICX

Overview

- C++20, based on AMReX and Kokkos
- core library
  - field containers/operations
  - particle containers/operations
  - interpolation
- models: variations of Vlasov-Maxwell
- steady rate of additions (numerical methods and models)

Progress

- checkpointing framework
- profiling/optimization
- infrastructure support
- refactoring
- GPU porting (ongoing)

GEMPICX

Checkpointing

- problem: 24h job duration limit
- solution: abstract "simulation" pattern implements generic checkpointing
- highlights:
  - global I/O registry
  - objects satisfy"I/O capable" C++20 concepts
  - generic "time stepping" implementation
  - user implements "initialize", "step" and "finalize"
  - user registers model objects with global I/O registry

Profiling/optimization

- AMReX offers "tiny profiler" for instrumentation
- NSIGHT Systems: particle loops dominate
- NSIGHT Compute: roofline result not ideal, but inconclusive
- idea: targeted loop optimization — moderate speedup
- idea: smaller block sizes — better speedup
- idea: move from `amrex::ParallelFor` to `Kokkos::parallel_for` — 10% to 15% speedup (for particles).

GEMPICX

Interpolation

- bottleneck identified with NSIGHT Compute
- task:
  - compute/store all stencil polynomials — 4 to 6 in each direction
  - compute products of stored values — different combinations for different derivatives
- idea: use one thread for each polynomial (fairly serious refactoring)
- status: separate proof-of-concept working on CPU, waiting for in-depth GPU performance analysis

Proof-of-concept

```
Kokkos::TeamPolicy<> policy(npoints, Kokkos::AUTO);
typedef Kokkos::TeamPolicy<>::member_type member_type;
Kokkos::parallel_for(
        policy,
        KOKKOS_LAMBDA(member_type team_member) {
    // get point
    const int p = team_member.league_rank();
    // get cell_id
    const int cc = std::floor(x[p] / grid_constant);
    const double xx = x[p] / grid_constant - cc;
    f[p] = 0;
    Kokkos::parallel_reduce(
        Kokkos::TeamThreadRange(team_member, stencil_size),
        [&] (const int& s, double& result) {
            // get location within stencil
            const int ii = 1 + s - stencil_size/2;
            result += (
                pp[(ii+stencil_size) % stencil_size](xx)
                * grid_data[(cc+grid_size+ii) % grid_size]);
            },
        f[p]);
```

GEMPICX final comments

Register pressure/kernel already highly optimized

	Duration [ms (1,108.35 m	Ru (70	Compute Throughput	Memory Throughput ['	# Registers [register/t	Grid Size	Block Size [block]	R
le....	343.54		75.81	24.19	224	256000, 1, 1	64, 1, 1	
le.....	334.94		77.75	23.28	224	256000, 1, 1	64, 1, 1	
le.....	213.04		33.75	32.68	184	64000, 1, 1	256, 1, 1	
le.....	198.72		36.18	31.94	184	64000, 1, 1	256, 1, 1	

Ongoing work

- Port existing CPU implementation of new model to GPU
- revisit profiling/optimization questions after separate numerical method implemented.

Overview

GRILLIX

- C++ extension proof of concept worked
- decided to refactor in C++ with Kokkos

GEMPICX

- performance analysis
- further optimization may need refactoring
- porting new code as it's coming in

THANK YOU