Blockstructured AMR

AMROC provides a generic interface to the most efficient adaptive method for hyperbolic equations

\[ \partial_t q(x, t) + \nabla \cdot f(q(x, t)) = s(q(x, t)) \]

on blockstructured grids: The Berger-Collela AMR algorithm.

- Discretization necessary only for a single logically rectangular grid
- Spatial and temporal refinement, no global time step restriction
- No neighboring cell information has to be stored
- Efficient cache reuse and vectorization possible
- Simple load balancing
  - Extension to non-Cartesian geometries is difficult
  - Cluster algorithm necessary for grid generation
  - Hanging nodes unavoidable and require special treatment
  - Complex implementation

In AMROC each AMR calculation involves three abstraction levels:
1. Specific application and numerical scheme.
2. AMROC (Adaptive Mesh Refinement in Object-oriented C++).
3. Parallel hierarchical data structures that employ the MPI-library.
   - Data follows “floor plan” of a single Grid Hierarchy.
   - Data of all levels resides on the same node \( \rightarrow \) Most AMR operations are strictly local.
   - Neighboring grids are synchronized transparently even over processor borders when boundary conditions are applied.
   - Distribution algorithm: Generalization of Hilbert’s space-filling curve.

**Benchmark Run 1: Point-explosion in 3D**

Timing results for the Sedov-like point-explosion benchmark (Euler equations) proposed at the AMR 2003 workshop in Chicago. AMROC was the only AMR framework participating in the benchmark session that was able to solve this problems with an effective resolution of \( 1024^3 \) on the given benchmark machine.

- 3D-Wave-Prop. Method
- Base grid \( 32^3 \)
- Refinement factor 2
- Grid generation efficiency 85%
- Proper nesting enforced
- Buffer of 1 cell

**Future Implementation Plans**

- Generic module to support the application of the Ghost Fluid Method at internal boundaries and optimal coupling to AMR (in collaboration with Patrick Hung).
- Python interface layer between AMROC level and application-specific objects (in collaboration with Julian Cummings).
- Seamless integration into the Virtual Test Facility.
  - Specification of an AMR calculation with a single Python script that selects the integrator at runtime and specifies initial and boundary conditions.
  - Regridding algorithm that scales well in parallel and performs better for a huge number of refinement grids.
- Hierarchical data structures:
  - Incorporation of additional parallelized partitioning algorithms (in collaboration with Manish Parashar).
  - Extension of GridFunctions to node- and edge-based storage to allow for staggered discretizations.
  - Usage of Blitz++ library for algebraic operations on subgrid arrays (in collaboration with Julian Cummings).
  - Code upgrade to the C++ standard (in collaboration with Julian Cummings).

**Future Research Plans**

- Adaptive multigrid method to support time-implicit discretizations (most parts of the framework can be reused, e.g. the stencil correction at coarse-fine interfaces).
- Comparison of different techniques for AMR to handle non-Cartesian boundaries (body-fitted grids, construction of cutted cells on the fly, etc.) with the Ghost Fluid Method.