Blockstructured Hyperbolic AMR with AMROC

Ralf Deiterding
Center for Adaced Computing Research

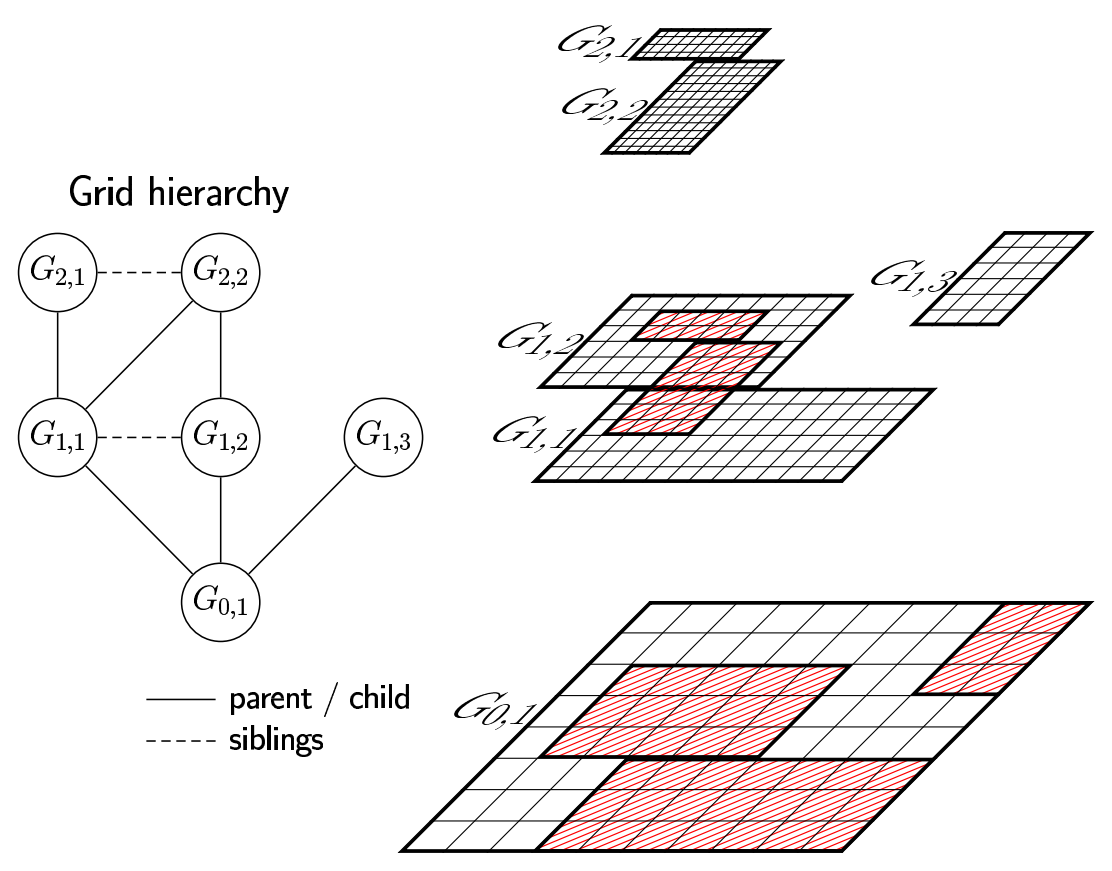
California Institute of Technology
1200 East California Blvd., Mail-Code 158-79
Pasadena, CA 91125
ralf@cacr.caltech.edu

Blockstructured AMR

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

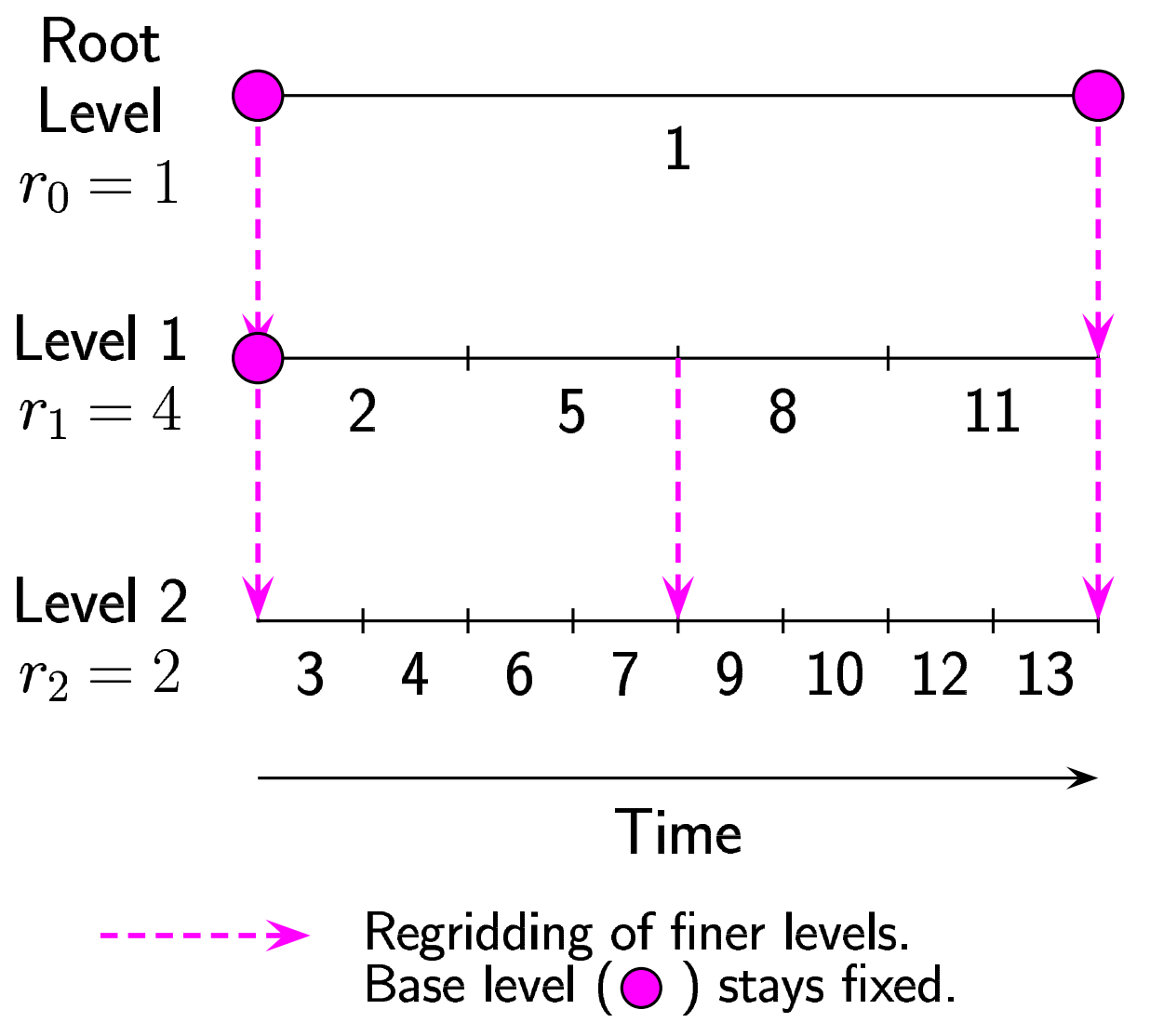
$$\partial_t \mathbf{q}(\mathbf{x},t) + \nabla \cdot \mathbf{f}(\mathbf{q}(\mathbf{x},t)) = \mathbf{s}(\mathbf{q}(\mathbf{x},t))$$

on blockstructured grids: The Berger-Collela AMR algorithm.



The blockstructured refinement strategy creates a hierarchy of properly nested subgrids.

+ 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



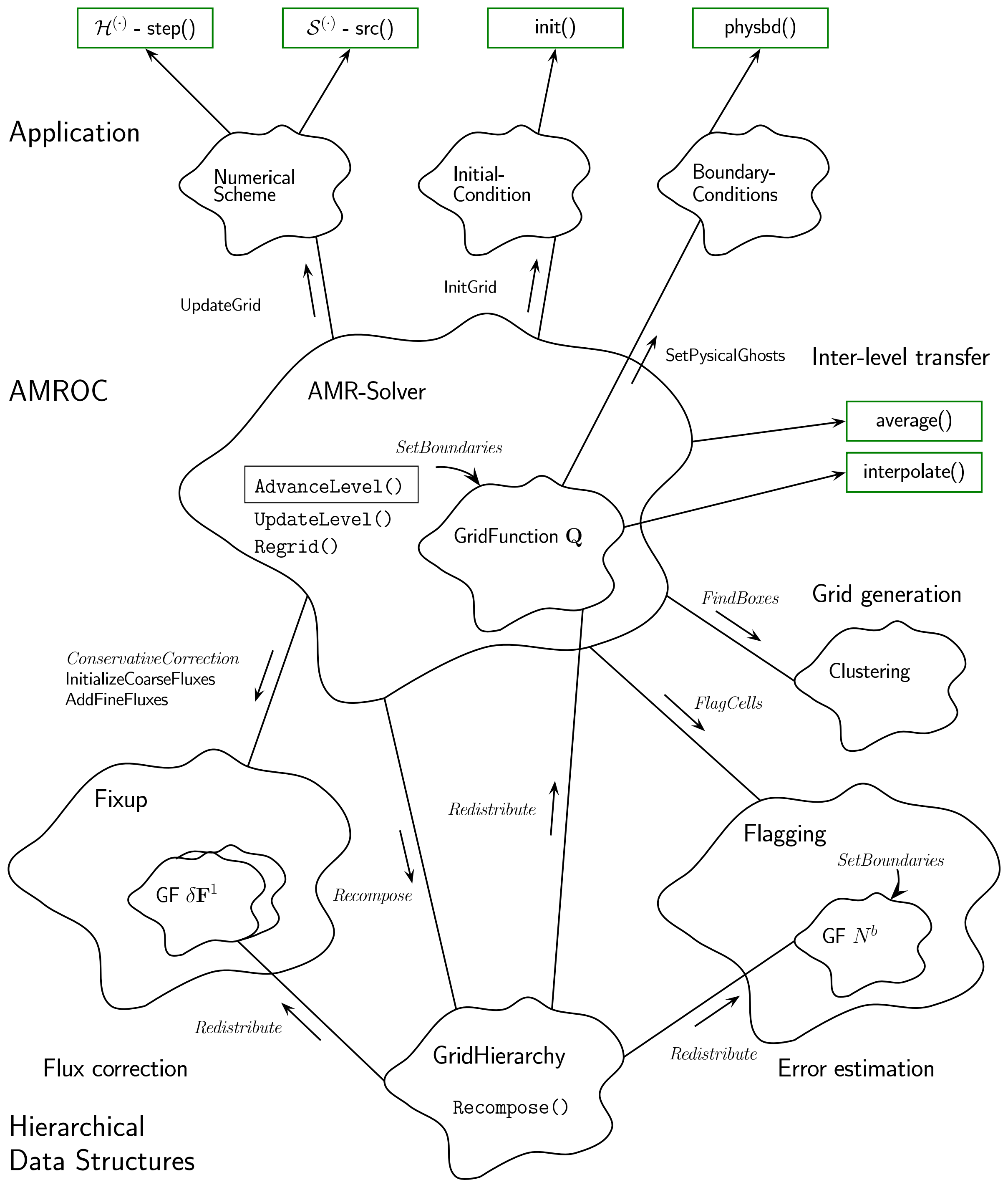
```
AdvanceLevel($l$)

Repeat $r_l$ times
  Set ghost cells of level $l$ at time $t$
  If (time to regrid?) Then
    Regrid($l$)
  Step $\Delta t_l$ on all grids at level $l$
  If (level $l+1$ exists?) Then
    Set level $l$ ghost cells at $t+\Delta t_l$
    AdvanceLevel($l+1$)
    Average level $l+1$ grids onto level $l$
```

AMR-algorithm and its recursive integration procedure. Refinement factor on level l is $r_l = \Delta t_{l-1}/\Delta t_l$.

Object-oriented Design

AMROC implements the AMR method in C++ and can be used directly in C++ or with Fortran functions called from C++ interface-objects.



Design of the AMROC framework. GF stands for GridFunction.

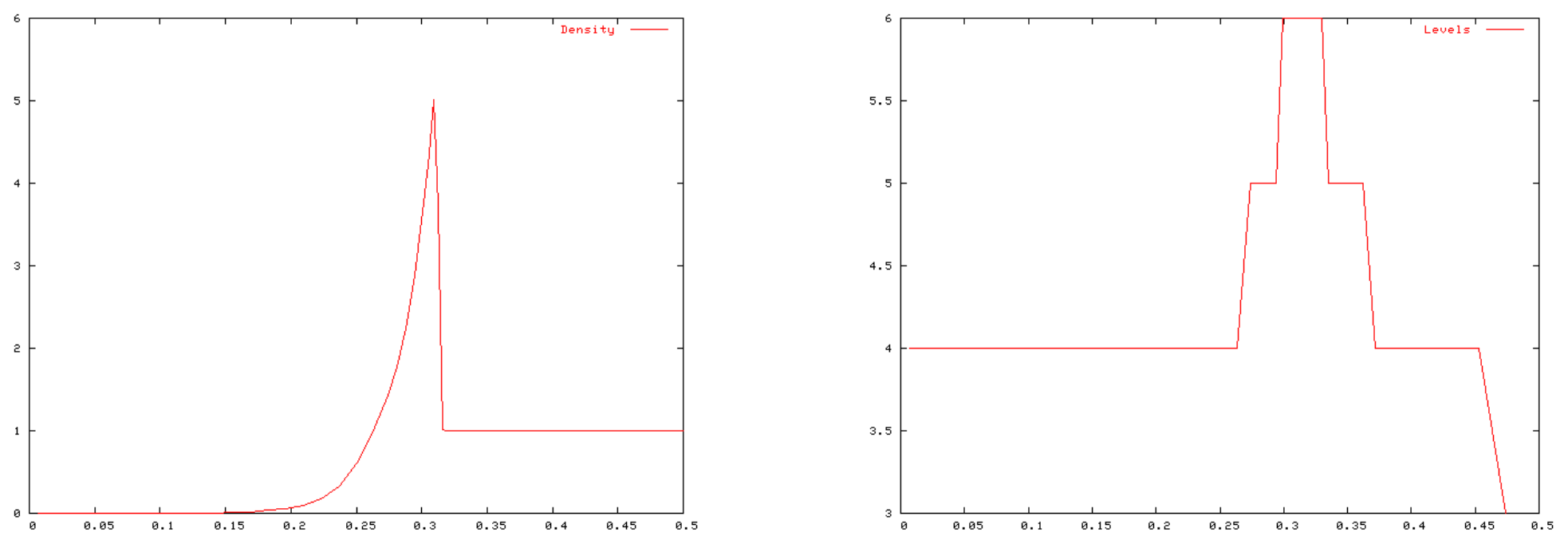
In AMROC each AMR calculation involves three abstraction levels:

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 → 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



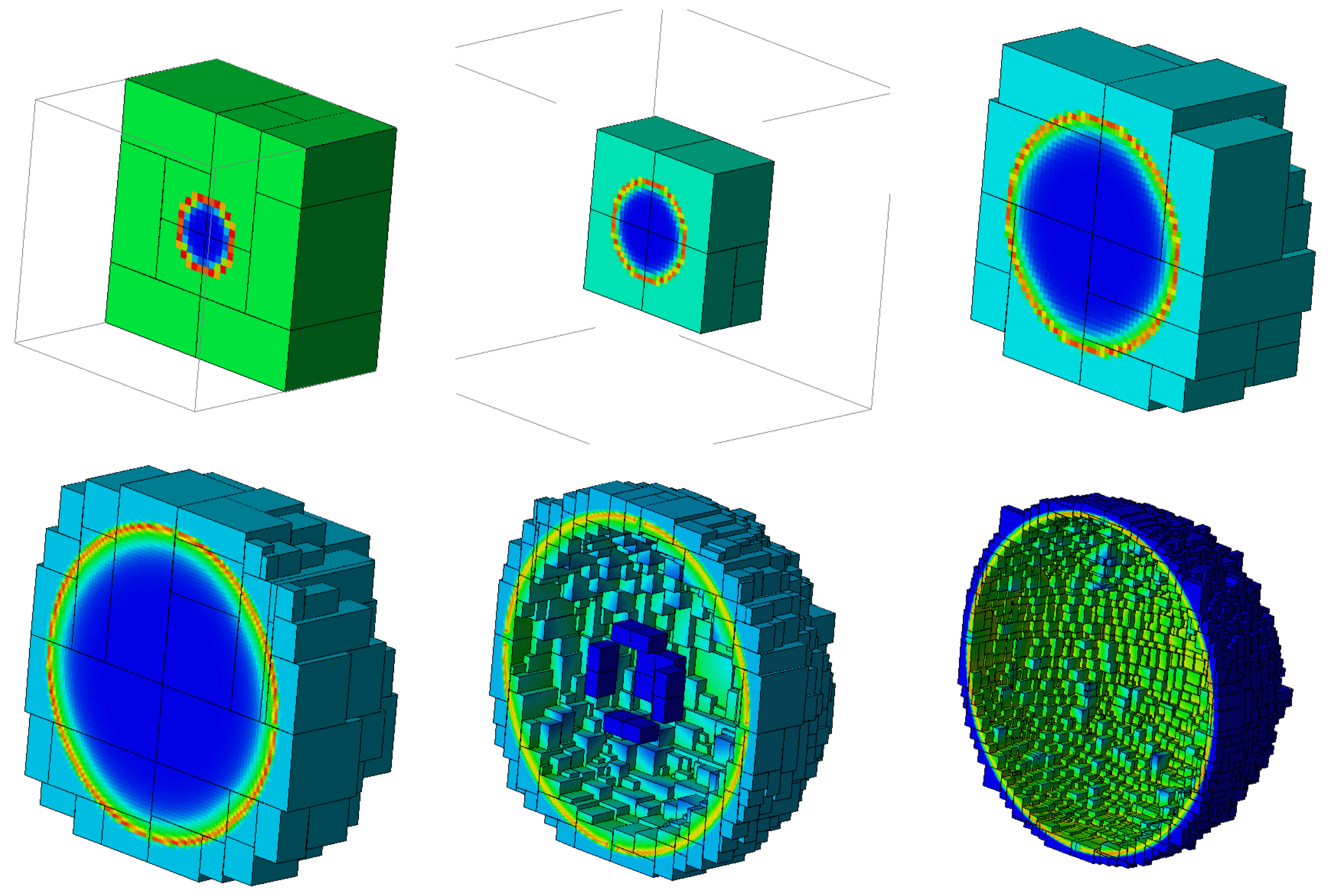
Cut through the symmetric solution for the effective resolution 1024^3.

l	$l_{\max} = 3$		$l_{\max} = 4$		$l_{\max} = 5$		$l_{\max} = 6$	
	Grids	Cells	Grids	Cells	Grids	Cells	Grids	Cells
1	28	32768	28	32768	33	32768	34	32768
2	8	32768	14	32768	20	32768	20	32768
3	63	115408	49	116920	43	125680	50	125144
4			324	398112	420	555744	193	572768
5					1405	1487312	1498	1795048
6							5266	5871128
$\sum$		180944		580568		2234272		8429624

Number of grids and cells.

Task [%]	$l_{\max} = 3$		$l_{\max} = 4$		$l_{\max} = 5$		$l_{\max} = 6$	
Integration	73.7		77.2		72.9		37.8	
Fixup	2.6	*46*	3.1	*58*	2.6	*42*	2.2	*45*
Boundary	10.1	*79*	6.3	*78*	5.1	*56*	6.9	*78*
Recomposition	7.4		8.0		15.1		50.4	
Clustering	0.5		0.6		0.7		1.0	
Output/Misc	5.7		4.0		3.6		1.7	
Time [min]	0.5		5.1		73.0		2100.0	
Uniform [min]	5.4		160		≈5000		≈180000.	
Factor of AMR savings	11		31		69		86	

Breakdown of CPU time on 8 nodes SGI Altix 3000 (Linux-based shared memory system).



Color plot of density on refinement grids for the 3D explosion problem on 6 levels.

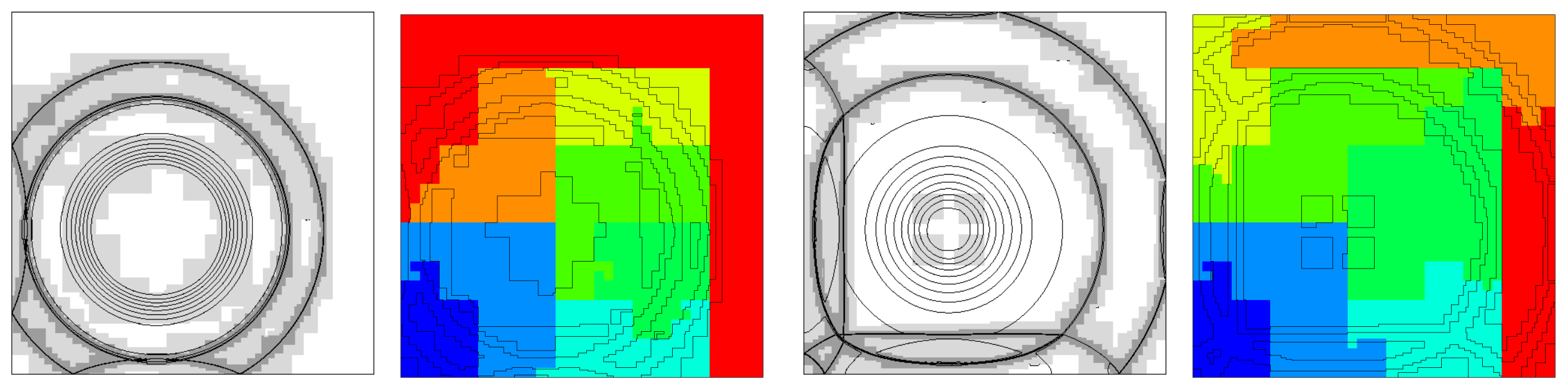
Benchmark Run 2: Blast Wave in 2D

A Circular Riemann problem for Euler equations expands in an enclosed box.

- 2D-Wave-Propagation Method with Roe's approximate solver
- Base grid 150 × 150
- 2 levels: factor 2, 4

Task [%]	P=1	P=2	P=4	P=8	P=16
Update by $\mathcal{H}^{(\cdot)}$	86.6	83.4	76.7	64.1	51.9
Flux correction	1.2	1.6	3.0	7.9	10.7
Boundary setting	3.5	5.7	10.1	15.6	18.3
Recomposition	5.5	6.1	7.4	9.9	14.0
Misc.	4.9	3.2	2.8	2.5	5.1
Time [min]	**151.9**	**79.2**	**43.4**	**23.3**	**13.9**
Efficiency [%]	**100.0**	**95.9**	**87.5**	**81.5**	**68.3**

Breakdown of CPU time on P nodes Pentium-III-1 GHz connected with Gigabit Ethernet (effective bandwidth ≈ 40 MB).



After 38 time steps

After 79 time steps

Isolines of density on two refinement levels (indicated by gray scales) and distribution to eight nodes (indicated by different colors).

Benchmark Rating

Both benchmarks used single-grid functions (finite volume scheme, interpolation, averaging) in Fortran 77.

+ The sequential computational performance of the AMROC framework in C++ is near to pure Fortran codes.
+ The performance of the MPI implementation on a shared memory machine is almost like in sequential.
+ At least a reasonable number of compute nodes of a distributed memory machine can be used efficiently.
- Benchmark 1 uncovers complexity problems in the regridding operation for a large numbers of refinement grids.

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).
  + Seamingless 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 subgrids.
- Hierarchical data structures:
  – Incorporation of additional parallelized partioning 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.