Lecture 1

Structured adaptive mesh refinement

Course *Block-structured Adaptive Mesh Refinement in C++*

Ralf Deiterding
University of Southampton
Engineering and the Environment
Highfield Campus, Southampton SO17 1BJ, UK
E-mail: r.deiterding@soton.ac.uk
Outline

Meshes and adaptation
Adaptivity on unstructured and structured meshes
Available SAMR software
Outline

**Meshes and adaptation**
Adaptivity on unstructured and structured meshes
Available SAMR software

**The serial Berger-Colella SAMR method**
Data structures and numerical update
Conservative flux correction
Level transfer operators
The basic recursive algorithm
Block generation and flagging of cells
Outline

Meshes and adaptation
   Adaptivity on unstructured and structured meshes
   Available SAMR software

The serial Berger-Colella SAMR method
   Data structures and numerical update
   Conservative flux correction
   Level transfer operators
   The basic recursive algorithm
   Block generation and flagging of cells

Parallel SAMR method
   Domain decomposition
   A parallel SAMR algorithm
Outline

**Meshes and adaptation**
- Adaptivity on unstructured and structured meshes
- Available SAMR software

**The serial Berger-Colella SAMR method**
- Data structures and numerical update
- Conservative flux correction
- Level transfer operators
- The basic recursive algorithm
- Block generation and flagging of cells

**Parallel SAMR method**
- Domain decomposition
- A parallel SAMR algorithm

**AMROC**
- Overview and basic software design
- Classes
Outline

**Meshes and adaptation**
Adaptivity on unstructured and structured meshes
Available SAMR software

**The serial Berger-Colella SAMR method**
Data structures and numerical update
Conservative flux correction
Level transfer operators
The basic recursive algorithm
Block generation and flagging of cells

**Parallel SAMR method**
Domain decomposition
A parallel SAMR algorithm

**AMROC**
Overview and basic software design
Classes
Adaptivity on unstructured and structured meshes

Elements of adaptive algorithms

- Base grid
Elements of adaptive algorithms

- Base grid
- Solver
Elements of adaptive algorithms

- Base grid
- Solver
- Error indicators
Elements of adaptive algorithms

- Base grid
- Solver
- Error indicators
- Grid manipulation
Elements of adaptive algorithms

- Base grid
- Solver
- Error indicators
- Grid manipulation
- Interpolation (restriction and prolongation)
Elements of adaptive algorithms

- Base grid
- Solver
- Error indicators
- Grid manipulation
- Interpolation (restriction and prolongation)
- Load-balancing
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones

![Diagram showing mesh refinement process](image-url)
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones

- Global time-step
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to be stored
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to be stored
  - Geometric flexible
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to be stored
  + Geometric flexible
  + No hanging nodes
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to stored
  + Geometric flexible
  + No hanging nodes
  + Easy to implement
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to stored
  + Geometric flexible
  + No hanging nodes
  + Easy to implement
  - Higher order difficult to achieve
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to be stored
  + Geometric flexible
  + No hanging nodes
  + Easy to implement
  - Higher order difficult to achieve
  - Cell aspect ratio must be considered
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to stored
  + Geometric flexible
  + No hanging nodes
  + Easy to implement
    - Higher order difficult to achieve
    - Cell aspect ratio must be considered
    - Fragmented data
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to stored
  + Geometric flexible
  + No hanging nodes
  + Easy to implement
  - Higher order difficult to achieve
  - Cell aspect ratio must be considered
  - Fragmented data
  - Cache-reuse / vectorization nearly impossible
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to be stored
  + Geometric flexible
  + No hanging nodes
  + Easy to implement
    - Higher order difficult to achieve
    - Cell aspect ratio must be considered
    - Fragmented data
    - Cache-reuse / vectorization nearly impossible
    - Complex load-balancing
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to stored
  + Geometric flexible
  + No hanging nodes
  + Easy to implement
  - Higher order difficult to achieve
  - Cell aspect ratio must be considered
  - Fragmented data
  - Cache-reuse / vectorization nearly impossible
  - Complex load-balancing
  - Complex synchronization
Structured mesh refinement techniques

▶ Block-based data of equal size
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
- Neighborhoods need not be stored
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
- Neighborhoods need not be stored

Wasted boundary space in a quad-tree
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
- Neighborhoods need not be stored

- Numerical scheme only for single regular block necessary

Wasted boundary space in a quad-tree
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
- Neighborhoods need not be stored
- Numerical scheme only for single regular block necessary
- Easy to implement
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
- Neighborhoods need not be stored
  - Numerical scheme only for single regular block necessary
  - Easy to implement
  - Simple load-balancing

Wasted boundary space in a quad-tree
Structured mesh refinement techniques

► Block-based data of equal size
► Block stored in a quad-tree
► Time-step refinement
► Global index coordinate system
► Neighborhoods need not be stored
+ Numerical scheme only for single regular block necessary
+ Easy to implement
+ Simple load-balancing
+ Parent/Child relations according to tree

Wasted boundary space in a quad-tree
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
- Neighborhoods need not be stored
- Numerical scheme only for single regular block necessary
- Easy to implement
- Simple load-balancing
- Parent/Child relations according to tree

/+/- Cache-reuse / vectorization only in data block

Wasted boundary space in a quad-tree
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
- Numerical scheme only for single patch necessary
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
- Numerical scheme only for single patch necessary
- Efficient cache-reuse / vectorization possible
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
  + Numerical scheme only for single patch necessary
  + Efficient cache-reuse / vectorization possible
  + Simple load-balancing
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
  + Numerical scheme only for single patch necessary
  + Efficient cache-reuse / vectorization possible
  + Simple load-balancing
  + Minimal synchronization overhead
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
- Numerical scheme only for single patch necessary
- Efficient cache-reuse / vectorization possible
- Simple load-balancing
- Minimal synchronization overhead
- Cells without mark are refined
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
- Numerical scheme only for single patch necessary
- Efficient cache-reuse / vectorization possible
- Simple load-balancing
- Minimal synchronization overhead
- Cells without mark are refined
- Hanging nodes unavoidable
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
  + Numerical scheme only for single patch necessary
  + Efficient cache-reuse / vectorization possible
  + Simple load-balancing
  + Minimal synchronization overhead
    - Cells without mark are refined
    - Hanging nodes unavoidable
    - Cluster-algorithm necessary
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system
- Numerical scheme only for single patch necessary
- Efficient cache-reuse / vectorization possible
- Simple load-balancing
- Minimal synchronization overhead
- Cells without mark are refined
- Hanging nodes unavoidable
- Cluster-algorithm necessary
- Difficult to implement
Simplified structured designs

Distributed memory parallelization fully supported if not otherwise stated.
Simplified structured designs

*Distributed memory parallelization fully supported if not otherwise stated.*

- **PARAMESH** (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)

- **Flash code** (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)

- **Uintah** (AMR code for simulation of accidental fires and explosions)
  - Only explicit algorithms considered
  - FSI coupling Material Point Method and ICE Method (Implicit, Continuous fluid, Eulerian)
  - [http://www.uintah.utah.edu](http://www.uintah.utah.edu)

- **DAGH/Grace** [Parashar and Browne, 1997]
  - Just C++ data structures but no methods
  - All grids are aligned to base mesh coarsened by factor 2
  - [http://userweb.cs.utexas.edu/users/dagh](http://userweb.cs.utexas.edu/users/dagh)
Simplified structured designs

*Distributed memory parallelization fully supported if not otherwise stated.*

- PARAMESH (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)

- Flash code (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)

- Uintah (AMR code for simulation of accidental fires and explosions)
  - Only explicit algorithms considered
  - FSI coupling Material Point Method and ICE Method (Implicit, Continuous fluid, Eulerian)
  - [http://www.uintah.utah.edu](http://www.uintah.utah.edu)

- DAGH/Grace [Parashar and Browne, 1997]
  - Just C++ data structures but no methods
  - All grids are aligned to bases mesh coarsened by factor 2
  - [http://userweb.cs.utexas.edu/users/dagh](http://userweb.cs.utexas.edu/users/dagh)
Simplified structured designs

*Distributed memory parallelization fully supported if not otherwise stated.*

- **PARAMESH** (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)
- **Flash code** (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)
- **Uintah** (AMR code for simulation of accidental fires and explosions)
  - Only explicit algorithms considered
  - FSI coupling Material Point Method and ICE Method (Implicit, Continuous fluid, Eulerian)
  - [http://www.uintah.utah.edu](http://www.uintah.utah.edu)
Simplified structured designs

Distributed memory parallelization fully supported if not otherwise stated.

- **PARAMESH** (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)

- **Flash code** (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)

- **Uintah** (AMR code for simulation of accidental fires and explosions)
  - Only explicit algorithms considered
  - FSI coupling Material Point Method and ICE Method (Implicit, Continuous fluid, Eulerian)
  - [http://www.uintah.utah.edu](http://www.uintah.utah.edu)

- **DAGH/Grace** [Parashar and Browne, 1997]
  - Just C++ data structures but no methods
  - All grids are aligned to bases mesh coarsened by factor 2
  - [http://userweb.cs.utexas.edu/users/dagh](http://userweb.cs.utexas.edu/users/dagh)
Systems that support general SAMR
**Systems that support general SAMR**

- **SAMRAI** - Structured Adaptive Mesh Refinement Application Infrastructure
  - Very mature SAMR system [Hornung et al., 2006]
  - Explicit algorithms directly supported, implicit methods through interface to Hypre package
  - Mapped geometry and some embedded boundary support
Systems that support general SAMR

- SAMRAI - Structured Adaptive Mesh Refinement Application Infrastructure
  - Very mature SAMR system [Hornung et al., 2006]
  - Explicit algorithms directly supported, implicit methods through interface to Hypre package
  - Mapped geometry and some embedded boundary support

- BoxLib, AmrLib, MGLib, HGProj
  - Berkley-Lab-AMR collection of C++ classes by J. Bell et al., 50,000 LOC [Rendleman et al., 2000]
  - Both multigrid and explicit algorithms supported
Available SAMR software

SAMRAI - Structured Adaptive Mesh Refinement Application Infrastructure

- Very mature SAMR system [Hornung et al., 2006]
- Explicit algorithms directly supported, implicit methods through interface to Hypre package
- Mapped geometry and some embedded boundary support

BoxLib, AmrLib, MGLib, HGProj

- Berkley-Lab-AMR collection of C++ classes by J. Bell et al., 50,000 LOC [Rendleman et al., 2000]
- Both multigrid and explicit algorithms supported

Chombo

- Redesign and extension of BoxLib by P. Colella et al.
- Both multigrid and explicit algorithms demonstrated
- Some embedded boundary support
- https://commons.lbl.gov/display/chombo
<table>
<thead>
<tr>
<th>Available SAMR software</th>
<th>Serial SAMR method</th>
<th>Parallel SAMR method</th>
<th>AMROC</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶️ Amrita by J. Quirk ▶️ Only 2D explicit finite volume methods supported ▶️ Embedded boundary algorithm ▶️ ▶️ ▶️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▶️ Cell-based Cartesian AMR: RAGE ▶️ Embedded boundary method ▶️ Explicit and implicit algorithms ▶️ [Gittings et al., 2008]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Further SAMR software
Further SAMR software

▶ Overture (Object-oriented tools for solving PDEs in complex geometries)
  ▶ Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  ▶ Explicit and implicit algorithms supported
  ▶ http://www.overtureframework.org
Further SAMR software

- **Overture** (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - [http://www.overtureframework.org](http://www.overtureframework.org)

- **AMRClaw within Clawpack** [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - [http://depts.washington.edu/clawpack](http://depts.washington.edu/clawpack)
Further SAMR software

- **Overture** (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - [http://www.overtureframework.org](http://www.overtureframework.org)

- **AMRClaw within Clawpack** [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - [http://depts.washington.edu/clawpack](http://depts.washington.edu/clawpack)

- **Amrita** by J. Quirk
  - Only 2D explicit finite volume methods supported
  - Embedded boundary algorithm
  - [http://www.amrita-cfd.org](http://www.amrita-cfd.org)
Further SAMR software

▶ Overture (Object-oriented tools for solving PDEs in complex geometries)
  ▶ Overlapping meshes for complex geometries by W. Henshaw et al.
    [Brown et al., 1997]
  ▶ Explicit and implicit algorithms supported
  ▶ http://www.overtureframework.org
▶ AMRClaw within Clawpack [Berger and LeVeque, 1998]
  ▶ Serial 2D Fortran 77 code for the explicit Wave Propagation method
    with own memory management
  ▶ http://depts.washington.edu/clawpack
▶ Amrita by J. Quirk
  ▶ Only 2D explicit finite volume methods supported
  ▶ Embedded boundary algorithm
  ▶ http://www.amrita-cfd.org
▶ Cell-based Cartesian AMR: RAGE
  ▶ Embedded boundary method
  ▶ Explicit and implicit algorithms
  ▶ [Gittings et al., 2008]
Outline

Meshes and adaptation
  Adaptivity on unstructured and structured meshes
  Available SAMR software

The serial Berger-Colella SAMR method
  Data structures and numerical update
  Conservative flux correction
  Level transfer operators
  The basic recursive algorithm
  Block generation and flagging of cells

Parallel SAMR method
  Domain decomposition
  A parallel SAMR algorithm

AMROC
  Overview and basic software design
  Classes
The $m$th refinement grid on level $l$

Notations:

- Boundary: $\partial G_{l,m}$
- Hull: $\bar{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$
- Ghost cell region: $\tilde{G}_{\sigma l,m} = G_{\sigma l,m} \setminus \bar{G}_{l,m}$

Interior grid with buffer cells - $G_{l,m}$
The $m$th refinement grid on level $l$

Notations:
- Boundary: $\partial G_{l,m}$
- Hull: $\bar{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$

Interior grid with buffer cells - $G_{l,m}$
The $m$th refinement grid on level $l$

Notations:

- Boundary: $\partial G_{l,m}$
- Hull: $\bar{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$

Interior grid with buffer cells - $G_{l,m}$
The $m$th refinement grid on level $l$ / 

**Notations:**

- **Boundary:** $\partial G_{l,m}$
- **Hull:** $\overline{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$

**Grids:**

- **Interior grid with buffer cells -** $G_{l,m}$
- **Complete grid with ghost cells -** $G_{\sigma l,m}$
The $m$th refinement grid on level $l$.

Notations:

- **Boundary**: $\partial G_{l,m}$
- **Hull**: $\bar{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$
- **Ghost cell region**: $\tilde{G}_{l,m}^\sigma = G_{l,m}^\sigma \setminus \bar{G}_{l,m}$
Refinement data

Resolution: $\Delta t_l := \frac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l}$
Refinement data

- Resolution: $\Delta t_l := \frac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l}$

- Refinement factor: $r_l \in \mathbb{N}, r_l \geq 2$ for $l > 0$ and $r_0 = 1$
Refinement data

- Resolution: \( \Delta t_l := \frac{\Delta t_{l-1}}{r_l} \) and \( \Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l} \)

- Refinement factor: \( r_l \in \mathbb{N}, r_l \geq 2 \) for \( l > 0 \) and \( r_0 = 1 \)

- Integer coordinate system for internal organization [Bell et al., 1994]:
  \[
  \Delta x_{n,l} \approx \mathop{\prod_{\kappa=l+1}^{l_{\text{max}}}} r_{\kappa}
  \]
Refinement data

▶ Resolution: $\Delta t_l := \frac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l}$

▶ Refinement factor: $r_l \in \mathbb{N}$, $r_l \geq 2$ for $l > 0$ and $r_0 = 1$

▶ Integer coordinate system for internal organization [Bell et al., 1994]:
\[
\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\text{max}}} r_{\kappa}
\]

▶ Computational Domain: $G_0 = \bigcup_{m=1}^{M_0} G_{0,m}$
Refinement data

- Resolution: \( \Delta t_l := \frac{\Delta t_{l-1}}{r_l} \) and \( \Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l} \)
- Refinement factor: \( r_l \in \mathbb{N}, r_l \geq 2 \) for \( l > 0 \) and \( r_0 = 1 \)
- Integer coordinate system for internal organization [Bell et al., 1994]:
  \[ \Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\text{max}}} r_{\kappa} \]
- Computational Domain: \( G_0 = \bigcup_{m=1}^{M_0} G_{0,m} \)
- Domain of level \( l \): \( G_l := \bigcup_{m=1}^{M_l} G_{l,m} \) with \( G_{l,m} \cap G_{l,n} = \emptyset \) for \( m \neq n \)
Refinement data

- Resolution: $\Delta t_l := \frac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l}$
- Refinement factor: $r_l \in \mathbb{N}$, $r_l \geq 2$ for $l > 0$ and $r_0 = 1$
- Integer coordinate system for internal organization [Bell et al., 1994]:
  \[\Delta x_{n,l} \approx l_{\max} \prod_{\kappa=l+1}^{l_{\max}} r_{\kappa}\]
- Computational Domain: $G_0 = \bigcup_{m=1}^{M_0} G_{0,m}$
- Domain of level $l$: $G_l := \bigcup_{m=1}^{M_l} G_{l,m}$ with $G_{l,m} \cap G_{l,n} = \emptyset$ for $m \neq n$
- Refinements are properly nested: $G_{l+1} \subset G_{l-1}$
Refinement data

- Resolution: \( \Delta t_l := \frac{\Delta t_{l-1}}{r_l} \) and \( \Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l} \)
- Refinement factor: \( r_l \in \mathbb{N}, r_l \geq 2 \) for \( l > 0 \) and \( r_0 = 1 \)
- Integer coordinate system for internal organization [Bell et al., 1994]:
  \[
  \Delta x_{n,l} \simeq \prod_{\kappa=l+1}^{l_{\text{max}}} r_{\kappa}
  \]
- Computational Domain: \( G_0 = \bigcup_{m=1}^{M_0} G_{0,m} \)
- Domain of level \( l \): \( G_l := \bigcup_{m=1}^{M_l} G_{l,m} \) with \( G_{l,m} \cap G_{l,n} = \emptyset \) for \( m \neq n \)
- Refinements are properly nested: \( G_l^1 \subset G_{l-1} \)
- Assume a FD scheme with stencil radius \( s \). Necessary data:
Refinement data

► Resolution: $\Delta t_l := \frac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l}$

► Refinement factor: $r_l \in \mathbb{N}$, $r_l \geq 2$ for $l > 0$ and $r_0 = 1$

► Integer coordinate system for internal organization [Bell et al., 1994]:

$$\Delta x_{n,l} \approx \prod_{\kappa = l+1}^{l_{\max}} r_{\kappa}$$

► Computational Domain: $G_0 = \bigcup_{m=1}^{M_0} G_{0,m}$

► Domain of level $l$: $G_l := \bigcup_{m=1}^{M_l} G_{l,m}$ with $G_{l,m} \cap G_{l,n} = \emptyset$ for $m \neq n$

► Refinements are properly nested: $G_{l}^1 \subset G_{l-1}$

► Assume a FD scheme with stencil radius $s$. Necessary data:

► Vector of state: $Q^l := \bigcup_m Q(G_{l,m}^s)$
Refinement data

- Resolution: $\Delta t_l := \frac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l}$

- Refinement factor: $r_l \in \mathbb{N}$, $r_l \geq 2$ for $l > 0$ and $r_0 = 1$

- Integer coordinate system for internal organization [Bell et al., 1994]:
  \[ \Delta x_{n,l} \equiv \prod_{\kappa=l+1}^{l_{\text{max}}} r_\kappa \]

- Computational Domain: $G_0 = \bigcup_{m=1}^{M_0} G_{0,m}$

- Domain of level $l$: $G_l := \bigcup_{m=1}^{M_l} G_{l,m}$ with $G_{l,m} \cap G_{l,n} = \emptyset$ for $m \neq n$

- Refinements are properly nested: $G^1_l \subset G_{l-1}$

- Assume a FD scheme with stencil radius $s$. Necessary data:
  - Vector of state: $Q^l := \bigcup_m Q(G^s_{l,m})$
  - Numerical fluxes: $F^{n,l} := \bigcup_m F^n(\tilde{G}_{l,m})$
Refinement data

- Resolution: $\Delta t_l := \frac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \frac{\Delta x_{n,l-1}}{r_l}$
- Refinement factor: $r_l \in \mathbb{N}, r_l \geq 2$ for $l > 0$ and $r_0 = 1$
- Integer coordinate system for internal organization [Bell et al., 1994]:
  \[
  \Delta x_{n,l} \approx \prod_{\kappa=l+1}^{l_{\text{max}}} r_{\kappa}
  \]
- Computational Domain: $G_0 = \bigcup_{m=1}^{M_0} G_{0,m}$
- Domain of level $l$: $G_l := \bigcup_{m=1}^{M_l} G_{l,m}$ with $G_{l,m} \cap G_{l,n} = \emptyset$ for $m \neq n$
- Refinements are properly nested: $G_l^1 \subset G_{l-1}$
- Assume a FD scheme with stencil radius $s$. Necessary data:
  - Vector of state: $Q^l := \bigcup_m Q(G_{l,m}^s)$
  - Numerical fluxes: $F^{n,l} := \bigcup_m F^n(\bar{G}_{l,m})$
  - Flux corrections: $\delta F^{n,l} := \bigcup_m \delta F^n(\partial G_{l,m})$
Setting of ghost cells
Setting of ghost cells
Setting of ghost cells

\[
\text{Synchronization with } G_l - \tilde{S}_{l,m}^s = \tilde{G}_{l,m}^s \cap G_l
\]
Setting of ghost cells

Synchronization with $G_l - \tilde{S}_{l,m}^s = \tilde{G}_{l,m}^s \cap G_l$

Physical boundary conditions - $\tilde{P}_{l,m}^s = \tilde{G}_{l,m}^s \setminus G_0$
Setting of ghost cells

Synchronization with $G_l - \tilde{S}_{l,m}^s = \tilde{G}_{l,m}^s \cap G_l$

Physical boundary conditions - $\tilde{P}_{l,m}^s = \tilde{G}_{l,m}^s \setminus G_0$

Interpolation from $G_{l-1} - \tilde{I}_{l,m}^s = \tilde{G}_{l,m}^s \setminus (\tilde{S}_{l,m}^s \cup \tilde{P}_{l,m}^s)$
Numerical update

Time-explicit conservative finite volume scheme

\[ \mathcal{H}^{(\Delta t)} : Q_{jk}(t+\Delta t) = Q_{jk}(t) - \frac{\Delta t}{\Delta x_1} \left( F_{j+\frac{1}{2},k}^1 - F_{j-\frac{1}{2},k}^1 \right) - \frac{\Delta t}{\Delta x_2} \left( F_{j,k+\frac{1}{2}}^2 - F_{j,k-\frac{1}{2}}^2 \right) \]
Numerical update

Time-explicit conservative finite volume scheme

\[ \mathcal{H}(\Delta t) : Q_{jk}(t+\Delta t) = Q_{jk}(t) - \frac{\Delta t}{\Delta x_1} \left( F^1_{j+\frac{1}{2},k} - F^1_{j-\frac{1}{2},k} \right) - \frac{\Delta t}{\Delta x_2} \left( F^2_{j,k+\frac{1}{2}} - F^2_{j,k-\frac{1}{2}} \right) \]

UpdateLevel(l)

For all \( m = 1 \) To \( M_l \) Do

\[ Q(G^{s}_{l,m}, t) \xrightarrow{\mathcal{H}(\Delta t_l)} Q(G_{l,m}, t + \Delta t_l), F^n(\bar{G}_{l,m}, t) \]
Numerical update

Time-explicit conservative finite volume scheme

\[ \mathcal{H}^{(\Delta t)} : Q_{jk}(t+\Delta t) = Q_{jk}(t) - \frac{\Delta t}{\Delta x_1} \left( F^1_{j+\frac{1}{2},k} - F^1_{j-\frac{1}{2},k} \right) - \frac{\Delta t}{\Delta x_2} \left( F^2_{j,k+\frac{1}{2}} - F^2_{j,k-\frac{1}{2}} \right) \]

UpdateLevel(l)

For all \( m = 1 \) To \( M_l \) Do

\[ Q(G^s_{l,m}, t) \xrightarrow{\mathcal{H}^{(\Delta t_l)}} Q(G_{l,m}, t + \Delta t_l) , F^n(\bar{G}_{l,m}, t) \]

If level \( l + 1 \) exists

Init \( \delta F^{n,l+1} \) with \( F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t) \)
Numerical update

Time-explicit conservative finite volume scheme

\[ \mathcal{H}^{(\Delta t)} : Q_{jk}(t+\Delta t) = Q_{jk}(t) - \frac{\Delta t}{\Delta x_1} \left( F_{j+\frac{1}{2},k}^1 - F_{j-\frac{1}{2},k}^1 \right) - \frac{\Delta t}{\Delta x_2} \left( F_{j,k+\frac{1}{2}}^2 - F_{j,k-\frac{1}{2}}^2 \right) \]

\text{UpdateLevel}(l)

For all \( m = 1 \) To \( M_l \) Do

\[ Q(G_{l,m}^s, t) \xrightarrow{\mathcal{H}^{(\Delta t_l)}} Q(G_{l,m}, t + \Delta t_l), F^n(\bar{G}_{l,m}, t) \]

If level \( l > 0 \)

Add \( F^n(\partial G_{l,m}, t) \) to \( \delta F^{n,l} \)

If level \( l + 1 \) exists

Init \( \delta F^{n,l+1} \) with \( F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t) \)
Conservative flux correction

Example: Cell $j, k$

\[
\dot{Q}^l_{jk}(t + \Delta t_l) = Q^l_{jk}(t) - \frac{\Delta t_l}{\Delta x_{1,l}} \left( F^{1,l}_{j+\frac{1}{2},k} - \frac{1}{r^2_{l+1}} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\iota=0}^{r_{l+1}-1} F^{1,l+1}_{v+\frac{1}{2},w+\iota} (t + \kappa \Delta t_{l+1}) \right) \\
- \frac{\Delta t_l}{\Delta x_{2,l}} \left( F^{2,l}_{j,k+\frac{1}{2}} - F^{2,l}_{j,k-\frac{1}{2}} \right)
\]

Correction pass:

\[
\begin{array}{c}
w \\
\downarrow \quad \downarrow \\
v \\
\downarrow \quad \downarrow \\
v+1 \\
\downarrow \quad \downarrow \\
\cdots \\
\end{array}
\]

\[
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]

\[
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]

\[
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]

\[
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]

\[
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]
Conservative flux correction

Example: Cell $j, k$

$$
\hat{Q}_{jk}^{l}(t + \Delta t) = Q_{jk}^{l}(t) - \frac{\Delta t}{\Delta x_{1,l}} \left( F_{j+\frac{1}{2},k}^{1,l} - \frac{1}{r_{l+1}^{2}} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\iota=0}^{r_{l+1}-1} F_{v+\frac{1}{2},w+\iota}^{1,l+1} (t + \kappa \Delta t_{l+1}) \right)
$$

$$
- \frac{\Delta t}{\Delta x_{2,l}} \left( F_{j,k+\frac{1}{2}}^{2,l} - F_{j,k-\frac{1}{2}}^{2,l} \right)
$$

Correction pass:

1. $\delta F_{j-\frac{1}{2},k}^{1,l+1} := -F_{j-\frac{1}{2},k}^{1,l}$
Conservative flux correction

Example: Cell $j, k$

$$
\dot{Q}_{jk}^l(t + \Delta t_l) = Q_{jk}^l(t) - \frac{\Delta t_l}{\Delta x_{1,l}} \left( F_{j+\frac{1}{2},k}^1,l - \frac{1}{r_{l+1}^2} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\iota=0}^{r_{l+1}-1} F_{v+\frac{1}{2},w+\iota}^{1,l+1} (t + \kappa \Delta t_{l+1}) \right)
- \frac{\Delta t_l}{\Delta x_{2,l}} \left( F_{j,k+\frac{1}{2}}^2,l - F_{j,k-\frac{1}{2}}^2,l \right)
$$

Correction pass:

1. $\delta F_{j-\frac{1}{2},k}^{1,l+1} := -F_{j-\frac{1}{2},k}^{1,l}$

2. $\delta F_{j-\frac{1}{2},k}^{1,l+1} := \delta F_{j-\frac{1}{2},k}^{1,l+1} + \frac{1}{r_{l+1}^2} \sum_{\iota=0}^{r_{l+1}-1} F_{v+\frac{1}{2},w+\iota}^{1,l+1} (t + \kappa \Delta t_{l+1})$
Conservative flux correction

Example: Cell $j, k$

$$
\bar{Q}_{jk}^l(t + \Delta t_l) = Q_{jk}^l(t) - \frac{\Delta t_l}{\Delta x_{1,l}} \left( F_{j+\frac{1}{2},k}^{1,l} - \frac{1}{r_{l+1}^2} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\iota=0}^{r_{l+1}-1} F_{v+\frac{1}{2},w+\iota}^{1,l+1} (t + \kappa \Delta t_{l+1}) \right) \\
- \frac{\Delta t_l}{\Delta x_{2,l}} \left( F_{j,k+\frac{1}{2}}^{2,l} - F_{j,k-\frac{1}{2}}^{2,l} \right)
$$

Correction pass:

1. $\delta F_{j-\frac{1}{2},k}^{1,l+1} := -F_{j-\frac{1}{2},k}^{1,l}$

2. $\delta F_{j-\frac{1}{2},k}^{1,l+1} := \delta F_{j-\frac{1}{2},k}^{1,l+1} + \frac{1}{r_{l+1}^2} \sum_{\iota=0}^{r_{l+1}-1} F_{v+\frac{1}{2},w+\iota}^{1,l+1} (t + \kappa \Delta t_{l+1})$

3. $\bar{Q}_{jk}^l(t + \Delta t_l) := Q_{jk}^l(t + \Delta t_l) + \frac{\Delta t_l}{\Delta x_{1,l}} \delta F_{j-\frac{1}{2},k}^{1,l+1}$
Conservative flux correction II

Conservative flux correction

\[ \text{ corrections } \delta F_{n, l+1} \text{ stored on level } l+1 \text{ along } \partial G_{l+1} \text{ (lower-dimensional data coarsened by } r_{l+1}) \]

\[ \text{ with level } l \text{ fluxes } F_{n, l}(\bar{G}_l \cap \partial G_{l+1}) \]

\[ \text{ Add level } l+1 \text{ fluxes } F_{n, l+1}(\partial G_{l+1}) \text{ to } \delta F_{n, l+1} \]
Conservative flux correction II

Level $l$ cells needing correction $(G_{l+1}^{r+1} \setminus G_{l+1}) \cap G_l$

- Cells to correct
Conservative flux correction II

Level $l$ cells needing correction $(G_{l+1}^{r_{l+1}} \setminus G_{l+1}) \cap G_l$

Corrections $\delta F^{n,l+1}$ stored on level $l + 1$ along $\partial G_{l+1}$ (lower-dimensional data coarsened by $r_{l+1}$)

- Cells to correct
- $\delta F^{n,l+1}$
Conservative flux correction II

- Level $l$ cells needing correction $(G_{l+1}^{r+1} \setminus G_{l+1}) \cap G_l$
- Corrections $\delta F^{n,l+1}$ stored on level $l+1$ along $\partial G_{l+1}$ (lower-dimensional data coarsened by $r_{l+1}$)
- Init $\delta F^{n,l+1}$ with level $l$ fluxes $F^{n,l}(\bar{G}_l \cap \partial G_{l+1})$
Conservative flux correction II

- Level $l$ cells needing correction $(G_{l+1}^{r+1} \setminus G_{l+1}) \cap G_l$
- Corrections $\delta F^{n,l+1}$ stored on level $l+1$ along $\partial G_{l+1}$ (lower-dimensional data coarsened by $r_{l+1}$)
- Init $\delta F^{n,l+1}$ with level $l$ fluxes $F^{n,l} (\tilde{G}_l \cap \partial G_{l+1})$
- Add level $l+1$ fluxes $F^{n,l+1} (\partial G_{l+1})$ to $\delta F^{n,l}$

- Cells to correct
- $F^{n,l}$
- $F^{n,l+1}$
- $\delta F^{n,l+1}$
Level transfer operators

Conservative averaging (restriction):
Replace cells on level \( l \) covered by level \( l + 1 \), i.e. \( G_l \cap G_{l+1} \), by

\[
\hat{Q}_{jk}^l := \frac{1}{(r_{l+1})^2} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\iota=0}^{r_{l+1}-1} Q_{v+\kappa, w+\iota}^{l+1}
\]

\[
Q_{\cdot k}^{l} = (x_{1,l}^{j-1}, x_{2,l}^{k-1})
\]

\[
Q_{\cdot v}^{l} = \cdot
\]

\[
Q_{w}^{l} = \cdot
\]

\[
Q_{\cdot \cdot}^{l} = \cdot
\]
Level transfer operators

Conservative averaging (restriction):
Replace cells on level $l$ covered by level $l+1$, i.e. $G_l \cap G_{l+1}$, by

$$
\hat{Q}_{jk}^l := \frac{1}{(r_{l+1})^2} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\nu=0}^{r_{l+1}-1} Q_{v+\kappa,w+\nu}^{l+1}
$$

Bilinear interpolation (prolongation):

$$
\hat{Q}_{vw}^{l+1} := (1 - f_1)(1 - f_2) Q_{j-1,k-1}^l + f_1(1 - f_2) Q_{j-1,k+1}^l +
(1 - f_1)f_2 Q_{j-1,k}^l + f_1f_2 Q_{jk}^l
$$

with factors $f_1 := \frac{x_{1,l+1} - x_{1,l}^{j-1}}{\Delta x_{1,l}}$, $f_2 := \frac{x_{2,l+1}^{w} - x_{2,l}^{k-1}}{\Delta x_{2,l}}$ derived from the spatial coordinates of the cell centers $(x_{1,l}^{j-1}, x_{2,l}^{k-1})$ and $(x_{1,l+1}, x_{2,l+1}^{w})$. 
Level transfer operators

Conservative averaging (restriction):
Replace cells on level $l$ covered by level $l + 1$, i.e. $G_l \cap G_{l+1}$, by

$$\hat{Q}_{jk}^l := \frac{1}{(r_{l+1})^2} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\iota=0}^{r_{l+1}-1} Q_{v+\kappa,w+\iota}^{l+1},$$

Bilinear interpolation (prolongation):

$$\tilde{Q}_{vw}^{l+1} := (1 - f_1) (1 - f_2) Q_{j-1,k-1}^l + f_1 (1 - f_2) Q_{j,k-1}^l + (1 - f_1) f_2 Q_{j-1,k}^l + f_1 f_2 Q_{jk}^l$$

with factors $f_1 := \frac{x_{v,1}^{l+1} - x_{j-1,1}^{l}}{\Delta x_{1,l}}$, $f_2 := \frac{x_{w,2}^{l+1} - x_{k-1,2}^{l}}{\Delta x_{2,l}}$ derived from the spatial coordinates of the cell centers $(x_{1,l}^{j-1}, x_{2,l}^{k-1})$ and $(x_{1,l+1}^{v}, x_{2,l+1}^{w})$.

For boundary conditions on $\tilde{I}_j$ : linear time interpolation

$$\tilde{Q}_{j}^{l+1}(t + \kappa \Delta t_{l+1}) := \left(1 - \frac{\kappa}{r_{l+1}}\right) \tilde{Q}_{j}^{l+1}(t) + \frac{\kappa}{r_{l+1}} \tilde{Q}_{j}^{l+1}(t + \Delta t_l) \quad \text{for } \kappa = 0, \ldots, r_{l+1}.$$
Recursive integration order

**Root Level**
- \( r_0 = 1 \)

**Level 1**
- \( r_1 = 4 \)

**Level 2**
- \( r_2 = 2 \)

- Regridding of finer levels.
- Base level (○) stays fixed.

Time

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>5</th>
<th>8</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recursive integration order

▶ Space-time interpolation of coarse data to set $l_i^s, l > 0$

Root Level
$r_0 = 1$

Level 1
$r_1 = 4$

Level 2
$r_2 = 2$

Time

Regridding of finer levels.
Base level (●) stays fixed.
Recursive integration order

- Space-time interpolation of coarse data to set $l_i^s$, $l > 0$
- Regridding:
  - Creation of new grids, copy existing cells on level $l > 0$

Root Level
\[ r_0 = 1 \]

Level 1
\[ r_1 = 4 \]

Level 2
\[ r_2 = 2 \]

Regridding of finer levels. Base level (●) stays fixed.
Recursive integration order

- Space-time interpolation of coarse data to set $I_i^s, i > 0$
- Regridding:
  - Creation of new grids, copy existing cells on level $i > 0$
  - Spatial interpolation to initialize new cells on level $i > 0$

Root Level
$r_0 = 1$

Level 1
$r_1 = 4$

Level 2
$r_2 = 2$

Regridding of finer levels.
Base level (●) stays fixed.
The basic recursive algorithm

\textbf{AdvanceLevel}(l)

Repeat \( r_l \) times
\hspace{1cm} Set ghost cells of \( Q^l(t) \)

\textbf{UpdateLevel}(l)

\( t := t + \Delta t_l \)
The basic recursive algorithm

**AdvanceLevel(l)**

Repeat \( r_l \) times
- Set ghost cells of \( Q^l(t) \)

**UpdateLevel(l)**
- If level \( l + 1 \) exists?
- Set ghost cells of \( Q^l(t + \Delta t_l) \)
- AdvanceLevel\((l + 1)\)

\[ t := t + \Delta t_l \]
The basic recursive algorithm

\textbf{AdvanceLevel}(l)

Repeat \( r_l \) times
- Set ghost cells of \( Q^l(t) \)

\textbf{UpdateLevel}(l)

If level \( l + 1 \) exists?
- Set ghost cells of \( Q^{l+1}(t + \Delta t_l) \)
- \textbf{AdvanceLevel}(l + 1)
- Average \( Q^{l+1}(t + \Delta t_l) \) onto \( Q^l(t + \Delta t_l) \)
- Correct \( Q^l(t + \Delta t_l) \) with \( \delta F^{l+1} \)

\( t := t + \Delta t_l \)
The basic recursive algorithm

AdvanceLevel($l$)

Repeat $r_l$ times
  Set ghost cells of $Q^l(t)$
  If time to regrid?
    Regrid($l$)
  UpdateLevel($l$)
  If level $l+1$ exists?
    Set ghost cells of $Q^{l+1}(t + \Delta t_l)$
    AdvanceLevel($l + 1$)
    Average $Q^{l+1}(t + \Delta t_l)$ onto $Q^l(t + \Delta t_l)$
    Correct $Q^l(t + \Delta t_l)$ with $\delta F^{l+1}$
  $t := t + \Delta t_l$

▶ Recursion
▶ Restriction and flux correction
▶ Re-organization of hierarchical data
The basic recursive algorithm

AdvanceLevel(/)

Repeat \( r \) times
    Set ghost cells of \( Q^l(t) \)
    If time to regrid?
        Regrid(/)
    UpdateLevel(/)
    If level \( l+1 \) exists?
        Set ghost cells of \( Q^{l+1}(t + \Delta t_l) \)
        AdvanceLevel(/ + 1)
        Average \( Q^{l+1}(t + \Delta t_l) \) onto \( Q^l(t + \Delta t_l) \)
        Correct \( Q^l(t + \Delta t_l) \) with \( \delta F^{l+1} \)

\( t := t + \Delta t_l \)

Start - Start integration on level 0

\( l = 0, r_0 = 1 \)
AdvanceLevel(/)
The basic recursive algorithm

\texttt{AdvanceLevel}(l)

Repeat \( r \) times
  Set ghost cells of \( Q^l(t) \)
  If time to regrid?
    \texttt{Regrid}(l)
  \texttt{UpdateLevel}(l)
  If level \( l + 1 \) exists?
    Set ghost cells of \( Q^{l+1}(t + \Delta t_l) \)
    \texttt{AdvanceLevel}(l + 1)
    Average \( Q^{l+1}(t + \Delta t_l) \) onto \( Q^l(t + \Delta t_l) \)
    Correct \( Q^l(t + \Delta t_l) \) with \( \delta F^{l+1} \)
    \( t := t + \Delta t_l \)

Start - Start integration on level 0

\( l = 0, \ r_0 = 1 \)
\texttt{AdvanceLevel}(l)

[Berger and Colella, 1988][Berger and Oliger, 1984]
Regridding algorithm

Regrid($l$) - Regrid all levels $i > l$

For $i = l_f$ DownTo $l$ Do
   Flag $N^i$ according to $Q^i(t)$
The basic recursive algorithm

Regridding algorithm

Regrid(l) - Regrid all levels \( l > l \)

For \( l = l_f \) Down to \( l \) Do

Flag \( N^l \) according to \( Q^l(t) \)

\[ N^l := \bigcup_m N(\partial G_{l,m}) \]
Regridding algorithm

Regrid($l$) - Regrid all levels $l > l$

For $l = l_f$ Down to $l$ Do
   Flag $N^l$ according to $Q^l(t)$
   If level $l + 1$ exists?
      Flag $N^l$ below $\tilde{G}^{l+2}$

- Refinement flags:
  $N^l := \bigcup_m N(\partial G_{l,m})$

- Activate flags below higher levels
Regridding algorithm

Regrid(l) - Regrid all levels $l > l$

For $l = l_f$ Down to $l$ Do
   Flag $N^l$ according to $Q^l(t)$
   If level $l + 1$ exists?
      Flag $N^l$ below $\tilde{G}^{l+2}$
      Flag buffer zone on $N^l$
   $\triangleright$ Refinement flags:
   $N^l := \bigcup_m N(\partial G_{l,m})$
   $\triangleright$ Activate flags below higher levels
   $\triangleright$ Flag buffer cells of $b > \kappa_r$ cells,
   $\kappa_r$ steps between calls of $\text{Regrid}(l)$

Structured adaptive mesh refinement
Regridding algorithm

Regrid($l$) - Regrid all levels $i > l$

For $i = l_f$ Down to $i$ Do
   Flag $N^i$ according to $Q^i(t)$
   If level $i + 1$ exists?
      Flag $N^i$ below $\tilde{G}^{i+2}$
   Flag buffer zone on $N^i$
   Generate $\tilde{G}^{i+1}$ from $N^i$

▶ Refinement flags:
   $N^l := \bigcup_m N(\partial G_{i,m})$

▶ Activate flags below higher levels

▶ Flag buffer cells of $b > \kappa_r$ cells,
   $\kappa_r$ steps between calls of
   Regrid($l$)

▶ Special cluster algorithm
Regridding algorithm

Regrid(l) – Regrid all levels $l > l$

For $l = l_f$ Down to $l$ Do
  Flag $N^l$ according to $Q^l(t)$
  If level $l + 1$ exists?
    Flag $N^l$ below $\tilde{G}^{l+2}$
    Flag buffer zone on $N^l$
    Generate $\tilde{G}^{l+1}$ from $N^l$

$\tilde{G}_l := G_l$

For $l = l$ To $l_f$ Do
  $C\tilde{G}_l := G_0 \setminus \tilde{G}_l$
  $\tilde{G}_{l+1} := \tilde{G}_{l+1} \setminus C\tilde{G}_l$

- Refinement flags: $N^l := \bigcup_m N(\partial G_{l,m})$
- Activate flags below higher levels
- Flag buffer cells of $b > \kappa_r$ cells, $\kappa_r$ steps between calls of Regrid(l)
- Special cluster algorithm
- Use complement operation to ensure proper nesting condition
Regridding algorithm

Regrid(\(l\)) - Regrid all levels \(l > l\)

For \(l = l_f\) Down to \(l\) Do
  Flag \(N^l\) according to \(Q^l(t)\)
  If level \(l + 1\) exists?
    Flag \(N^l\) below \(\tilde{G}^{l+2}\)
    Flag buffer zone on \(N^l\)
    Generate \(\tilde{G}^{l+1}\) from \(N^l\)

\(\tilde{G}_l := G_l\)

For \(l = l\) To \(l_f\) Do
  \(C\tilde{G}_l := G_0 \setminus \tilde{G}_l\)
  \(\tilde{G}_{l+1} := \tilde{G}_{l+1} \setminus C\tilde{G}_l\)

Recompose(\(l\))

- Refinement flags:
  \(N^l := \bigcup_m N(\partial G_{l,m})\)

- Activate flags below higher levels

- Flag buffer cells of \(b > \kappa_r\) cells,
  \(\kappa_r\) steps between calls of Regrid(\(l\))

- Special cluster algorithm

- Use complement operation to ensure proper nesting condition
Recomposition of data

Recompose(\(l\)) - Reorganize all levels \(l > l\)

For \(l = l + 1\) To \(l_f + 1\) Do

- Creates max. 1 level above \(l_f\), but can remove multiple level if \(\tilde{G}_l\) empty (no coarsening!)
Recomposition of data

Recompose(\(l\)) - Reorganize all levels \(l > l\)

For \(l = l + 1\) To \(l_f + 1\) Do

Interpolate \(Q^{l-1}(t)\) onto \(Q^l(t)\)

- Creates max. 1 level above \(l_f\), but can remove multiple level if \(\tilde{G}_l\) empty (no coarsening!)
- Use spatial interpolation on entire data \(\tilde{Q}^l(t)\)
Recomposition of data

Recompose($l$) – Reorganize all levels $i > l$

For $i = l + 1$ To $l_f + 1$ Do

- Interpolate $Q^{i-1}(t)$ onto $\tilde{Q}^i(t)$
- Copy $Q^i(t)$ onto $\tilde{Q}^i(t)$

- Creates max. 1 level above $l_f$, but can remove multiple level if $\tilde{G}_i$ empty (no coarsening!)
- Use spatial interpolation on entire data $\tilde{Q}^i(t)$
- Overwrite where old data exists
Recomposition of data

Recompose(/) - Reorganize all levels \( i > l \)

For \( i = l + 1 \) To \( l_f + 1 \) Do

- Interpolate \( Q^{i-1}(t) \) onto \( \tilde{Q}^l(t) \)
- Copy \( Q^l(t) \) onto \( \tilde{Q}^l(t) \)
- Set ghost cells of \( \tilde{Q}^l(t) \)

- Creates max. 1 level above \( l_f \), but can remove multiple level if \( \tilde{G}_l \) empty (no coarsening!)
- Use spatial interpolation on entire data \( \tilde{Q}^l(t) \)
- Overwrite where old data exists
- Synchronization and physical boundary conditions
Recomposition of data

Recompose($l$) – Reorganize all levels $i > l$

For $i = l + 1$ To $l_f + 1$ Do

Interpolate $Q^{i-1}(t)$ onto $\tilde{Q}^i(t)$
Copy $Q^i(t)$ onto $\tilde{Q}^i(t)$
Set ghost cells of $\tilde{Q}^i(t)$
$Q^i(t) := \tilde{Q}^i(t)$, $G_i := \tilde{G}_i$

- Creates max. 1 level above $l_f$, but can remove multiple level if $\tilde{G}_i$ empty (no coarsening!)
- Use spatial interpolation on entire data $\tilde{Q}^i(t)$
- Overwrite where old data exists
- Synchronization and physical boundary conditions
### Clustering by signatures

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| τ   | 6 | 6 | 2 | 3 | 2 | 2 | 2 | 2 |

- τ: Flagged cells per row/column
- Δ: Second derivative of τ, \( Δ = τ_{ν+1} - 2τ_ν + τ_{ν-1} \)

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]
Clustering by signatures

\[ \nu \quad \text{Flagged cells per row/column} \]
\[ \Delta \quad \text{Second derivative of } \nu, \quad \Delta = \nu_{\nu+1} - 2 \nu_\nu + \nu_{\nu-1} \]

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]
### Clustering by signatures

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>6</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Τ</th>
<th>6</th>
<th>6</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Δ</th>
<th>-2</th>
<th>3</th>
<th>-2</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|  Τ | Flagged cells per row/column |
| Δ  | Second derivative of Τ, Δ = Τ\(_{\nu+1}\) - 2 Τ\(_\nu\) + Τ\(_{\nu-1}\) |

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]
Clustering by signatures

\[ \begin{array}{cccccc}
6 & 6 & 6 & 2 & 3 & 2 \\
3 & 3 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 2 & 1 & 0 \\
\end{array} \]

\[ \begin{array}{cccccc}
6 & 6 & 6 & 2 & 3 & 2 \\
3 & 3 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 2 & 1 & 0 \\
\end{array} \]

\[ \begin{array}{cccccc}
\tau & \text{Flagged cells per row/column} \\
\Delta & \text{Second derivative of } \tau, \Delta = \tau_{\nu+1} - 2\tau_\nu + \tau_{\nu-1} \\
\end{array} \]

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]
### Recursive generation of $G_{l,m}$

1. 0 in $\bar{\gamma}$
2. Largest difference in $\Delta$
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$
Recursive generation of $\tilde{G}_{l,m}$

1. 0 in $\Upsilon$
2. Largest difference in $\Delta$
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$
Recursive generation of $\tilde{G}_{l,m}$

1. 0 in $\Upsilon$
2. Largest difference in $\Delta$
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$
Structured adaptive mesh refinement

Recursive generation of $\tilde{G}_{l,m}$

1. 0 in $\gamma$

2. Largest difference in $\Delta$

3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$
Refinement criteria

Scaled gradient of scalar quantity $w$

$$|w(Q_{j+1,k}) - w(Q_{jk})| > \varepsilon_w, \quad |w(Q_{j,k+1}) - w(Q_{jk})| > \varepsilon_w, \quad |w(Q_{j+1,k+1}) - w(Q_{jk})| > \varepsilon_w$$
Refinement criteria

Scaled gradient of scalar quantity $w$

$$|w(Q_{j+1,k}) - w(Q_{jk})| > \epsilon_w, \ |w(Q_{j,k+1}) - w(Q_{jk})| > \epsilon_w, \ |w(Q_{j+1,k+1}) - w(Q_{jk})| > \epsilon_w$$

Heuristic error estimation [Berger, 1982]:
Local truncation error of scheme of order $o$

$$q(x, t + \Delta t) - \mathcal{H}^{(\Delta t)}(q(\cdot, t)) = C\Delta t^{o+1} + O(\Delta t^{o+2})$$
Refinement criteria

Scaled gradient of scalar quantity $w$

$$|w(Q_{j+1,k}) - w(Q_{jk})| > \epsilon_w, \ |w(Q_{j,k+1}) - w(Q_{jk})| > \epsilon_w, \ |w(Q_{j+1,k+1}) - w(Q_{jk})| > \epsilon_w$$

Heuristic error estimation [Berger, 1982]:
Local truncation error of scheme of order $o$

$$q(x, t + \Delta t) - \mathcal{H}^{(\Delta t)}(q(\cdot, t)) = C\Delta t^{o+1} + O(\Delta t^{o+2})$$

For $q$ smooth after 2 steps $\Delta t$

$$q(x, t + \Delta t) - \mathcal{H}^{(\Delta t)}_2(q(\cdot, t - \Delta t)) = 2C\Delta t^{o+1} + O(\Delta t^{o+2})$$
Refinement criteria

Scaled gradient of scalar quantity $w$

$$|w(Q_{j+1,k}) - w(Q_{jk})| > \epsilon_w, |w(Q_{j,k+1}) - w(Q_{jk})| > \epsilon_w, |w(Q_{j+1,k+1}) - w(Q_{jk})| > \epsilon_w$$

Heuristic error estimation [Berger, 1982]:
Local truncation error of scheme of order $o$

$$q(x, t + \Delta t) - H^{(\Delta t)}(q(\cdot, t)) = C \Delta t^{o+1} + O(\Delta t^{o+2})$$

For $q$ smooth after 2 steps $\Delta t$

$$q(x, t + \Delta t) - H_2^{(\Delta t)}(q(\cdot, t - \Delta t)) = 2 C \Delta t^{o+1} + O(\Delta t^{o+2})$$

and after 1 step with $2\Delta t$

$$q(x, t + \Delta t) - H^{(2\Delta t)}(q(\cdot, t - \Delta t)) = 2^{o+1} C \Delta t^{o+1} + O(\Delta t^{o+2})$$
Refinement criteria

Scaled gradient of scalar quantity $w$

$$|w(Q_{j+1,k}) - w(Q_{jk})| > \epsilon_w, \ |w(Q_{j,k+1}) - w(Q_{jk})| > \epsilon_w, \ |w(Q_{j+1,k+1}) - w(Q_{jk})| > \epsilon_w$$

Heuristic error estimation [Berger, 1982]:

Local truncation error of scheme of order $o$

$$q(x, t + \Delta t) - \mathcal{H}(\Delta t)(q(\cdot, t)) = C\Delta t^{o+1} + O(\Delta t^{o+2})$$

For $q$ smooth after 2 steps $\Delta t$

$$q(x, t + \Delta t) - \mathcal{H}_2(\Delta t)(q(\cdot, t - \Delta t)) = 2C\Delta t^{o+1} + O(\Delta t^{o+2})$$

and after 1 step with $2\Delta t$

$$q(x, t + \Delta t) - \mathcal{H}(2\Delta t)(q(\cdot, t - \Delta t)) = 2^{o+1}C\Delta t^{o+1} + O(\Delta t^{o+2})$$

Gives

$$\mathcal{H}_2(\Delta t)(q(\cdot, t - \Delta t)) - \mathcal{H}(2\Delta t)(q(\cdot, t - \Delta t)) = (2^{o+1} - 2)C\Delta t^{o+1} + O(\Delta t^{o+2})$$
Heuristic error estimation for FV methods

1. Error estimation on interior cells
Heuristic error estimation for FV methods

1. Error estimation on interior cells

$H^{\Delta t_l} Q'(t_l - \Delta t_l)$
Heuristic error estimation for FV methods

1. Error estimation on interior cells

\[ H^{\Delta t_l} Q^l(t_l - \Delta t_l) \]
Heuristic error estimation for FV methods

1. Error estimation on interior cells

\[ \mathcal{H}^{\Delta t_l} Q_l(t_l - \Delta t_l) = \mathcal{H}^{\Delta t_l} (\mathcal{H}^{\Delta t_l} Q_l(t_l - \Delta t_l)) = \mathcal{H}_2^{\Delta t_l} Q_l(t_l - \Delta t_l) \]
Heuristic error estimation for FV methods

1. Error estimation on interior cells
   
   \( H_1^{\Delta t_l} Q^l(t_l - \Delta t_l) \)

2. Create temporary Grid coarsened by factor 2
   Initialize with fine-grid values of preceding time step

\[
H_1^{\Delta t_l} Q^l(t_l - \Delta t_l) = H_2^{\Delta t_l} Q^l(t_l - \Delta t_l)
\]
Heuristic error estimation for FV methods

1. Error estimation on interior cells

2. Create temporary Grid coarsened by factor 2
   Initialize with fine-grid-values of preceding time step

\[ H^{\Delta t_l} Q^l(t_l - \Delta t_l) = H^{\Delta t_l} (H^{\Delta t_l} Q^l(t_l - \Delta t_l)) = H_2^{\Delta t_l} Q^l(t_l - \Delta t_l) \]

\[ H^{2\Delta t_l} \bar{Q}^l(t_l - \Delta t_l) \]
Heuristic error estimation for FV methods

1. Error estimation on interior cells
2. Create temporary Grid coarsened by factor 2
   Initialize with fine-grid-values of preceding time step
3. Compare temporary solutions

\[ H^{\Delta t_l} Q^l(t_l - \Delta t_l) \]
\[ = H^{\Delta t_l} (H^{\Delta t_l} Q^l(t_l - \Delta t_l)) \]
\[ = H^{2\Delta t_l} Q^l(t_l - \Delta t_l) \]
\[ = H^{2\Delta t_l} \bar{Q}^l(t_l - \Delta t_l) \]
Usage of heuristic error estimation

Current solution integrated tentatively 1 step with $\Delta t_l$ and coarsened

$$\bar{Q}(t_l + \Delta t_l) := \text{Restrict} \left( H^{\Delta t_l} Q^{l}(t_l - \Delta t_l) \right)$$

Previous solution coarsened and integrated 1 step with $2\Delta t_l$

$$Q(t_l + \Delta t_l) := H^{2\Delta t_l} \text{Restrict} \left( Q^{l}(t_l - \Delta t_l) \right)$$
Usage of heuristic error estimation

Current solution integrated tentatively 1 step with $\Delta t_l$ and coarsened

$$\bar{Q}(t_l + \Delta t_l) : = \text{Restrict} \left( \mathcal{H}_2^{\Delta t_l} Q^l(t_l - \Delta t_l) \right)$$

Previous solution coarsened and integrated 1 step with $2\Delta t_l$

$$Q(t_l + \Delta t_l) : = \mathcal{H}_2^{2\Delta t_l} \text{Restrict} \left( Q^l(t_l - \Delta t_l) \right)$$

Local error estimation of scalar quantity $w$

$$\tau_{jk}^w : = \frac{|w(\bar{Q}_{jk}(t + \Delta t)) - w(Q_{jk}(t + \Delta t))|}{2^{o+1} - 2}$$
Usage of heuristic error estimation

Current solution integrated tentatively 1 step with $\Delta t_l$ and coarsened

$$\tilde{Q}(t_l + \Delta t_l) := \text{Restrict} \left( \mathcal{H}^{\Delta t_l} Q^l(t_l - \Delta t_l) \right)$$

Previous solution coarsened and integrated 1 step with $2\Delta t_l$

$$Q(t_l + \Delta t_l) := \mathcal{H}^{2\Delta t_l} \text{Restrict} \left( Q^l(t_l - \Delta t_l) \right)$$

Local error estimation of scalar quantity $w$

$$\tau_{jk}^w := \frac{|w(\tilde{Q}_{jk}(t + \Delta t)) - w(Q_{jk}(t + \Delta t))|}{2^{o+1} - 2}$$

In practice [Deiterding, 2003] use

$$\frac{\tau_{jk}^w}{\max(|w(Q_{jk}(t + \Delta t))|, S_w)} > \eta_r^w$$
Outline

Meshes and adaptation
  Adaptivity on unstructured and structured meshes
  Available SAMR software

The serial Berger-Colella SAMR method
  Data structures and numerical update
  Conservative flux correction
  Level transfer operators
  The basic recursive algorithm
  Block generation and flagging of cells

Parallel SAMR method
  Domain decomposition
  A parallel SAMR algorithm

AMROC
  Overview and basic software design
  Classes
Parallelization strategies

Decomposition of the hierarchical data

- Distribution of each grid
Parallelization strategies

Decomposition of the hierarchical data

- Distribution of each grid
- Separate distribution of each level, cf. [Rendleman et al., 2000]
Parallelization strategies

Decomposition of the hierarchical data

► Distribution of each grid
► Separate distribution of each level, cf. [Rendleman et al., 2000]
► Rigorous domain decomposition
Parallelization strategies

Decomposition of the hierarchical data

▷ Distribution of each grid

▷ Separate distribution of each level, cf. [Rendleman et al., 2000]

▷ Rigorous domain decomposition
  ▷ Data of all levels resides on same node
Parallelization strategies

Decomposition of the hierarchical data

- Distribution of each grid
- Separate distribution of each level, cf. [Rendleman et al., 2000]
- Rigorous domain decomposition
  - Data of all levels resides on same node
  - Grid hierarchy defines unique "floor-plan"
Parallelization strategies

Decomposition of the hierarchical data

- Distribution of each grid
- Separate distribution of each level, cf. [Rendleman et al., 2000]
- Rigorous domain decomposition
  - Data of all levels resides on same node
  - Grid hierarchy defines unique "floor-plan"
  - Redistribution of data blocks during reorganization of hierarchical data
Parallelization strategies

Decomposition of the hierarchical data

- Distribution of each grid
- Separate distribution of each level, cf. [Rendleman et al., 2000]
- Rigorous domain decomposition
  - Data of all levels resides on same node
  - Grid hierarchy defines unique "floor-plan"
  - Redistribution of data blocks during reorganization of hierarchical data
  - Synchronization when setting ghost cells
Rigorous domain decomposition formalized

Parallel machine with $P$ identical nodes. $P$ non-overlapping portions $G_0^p$, $p = 1, \ldots, P$ as

$$G_0 = \bigcup_{p=1}^{P} G_0^p \quad \text{with} \quad G_0^p \cap G_0^q = \emptyset \quad \text{for} \quad p \neq q$$
Rigorous domain decomposition formalized

Parallel machine with $P$ identical nodes. $P$ non-overlapping portions $G_0^p$, $p = 1, \ldots, P$ as

$$G_0 = \bigcup_{p=1}^{P} G_0^p \quad \text{with} \quad G_0^p \cap G_0^q = \emptyset \quad \text{for} \quad p \neq q$$

Higher level domains $G_l$ follow decomposition of root level

$$G_l^p := G_l \cap G_0^p$$
Rigorous domain decomposition formalized

Parallel machine with $P$ identical nodes. $P$ non-overlapping portions $G_0^p$, $p = 1, \ldots, P$ as

$$G_0 = \bigcup_{p=1}^{P} G_0^p \quad \text{with} \quad G_0^p \cap G_0^q = \emptyset \quad \text{for} \quad p \neq q$$

Higher level domains $G_l$ follow decomposition of root level

$$G_l^p := G_l \cap G_0^p$$

With $\mathcal{N}_l(\cdot)$ denoting number of cells, we estimate the workload as

$$W(\Omega) = \sum_{l=0}^{l_{\max}} \left[ \mathcal{N}_l(G_l \cap \Omega) \prod_{\kappa=0}^{l} r_\kappa \right]$$
Rigorous domain decomposition formalized

Parallel machine with $P$ identical nodes. $P$ non-overlapping portions $G_0^p$, $p = 1, \ldots, P$ as

$$G_0 = \bigcup_{p=1}^{P} G_0^p \quad \text{with} \quad G_0^p \cap G_0^q = \emptyset \quad \text{for} \ p \neq q$$

Higher level domains $G_l$ follow decomposition of root level

$$G_l^p := G_l \cap G_0^p$$

With $N_l(\cdot)$ denoting number of cells, we estimate the workload as

$$\mathcal{W}(\Omega) = \sum_{l=0}^{l_{\text{max}}} \left[ N_l(G_l \cap \Omega) \prod_{\kappa=0}^{l} r_\kappa \right]$$

Equal work distribution necessitates

$$\mathcal{L}^p := \frac{P \cdot \mathcal{W}(G_0^p)}{\mathcal{W}(G_0)} \approx 1 \quad \text{for all} \ p = 1, \ldots, P$$

[Deiterding, 2005]
Ghost cell setting

Processor 1      Processor 2

Ghost cell values:
- Yellow: Interpolation
- Green: Local synchronization
- Pink: Parallel synchronization
- Blue: Physical boundary

Structured adaptive mesh refinement
Ghost cell setting

Ghost cell values:
- Yellow: Interpolation
- Green: Local synchronization
- Pink: Parallel synchronization
- Blue: Physical boundary
Ghost cell setting

Local synchronization

\[ \tilde{S}_{l,m}^{s,p} = \tilde{G}_{l,m}^{s,p} \cap G^p \]

Ghost cell values:
- Interpolation
- Local synchronization
- Parallel synchronization
- Physical boundary
Ghost cell setting

Local synchronization

\[ \tilde{S}^s_{l,m} = \tilde{G}^s_{l,m} \cap G^p_l \]

Parallel synchronization

\[ \tilde{S}^s_{l,m} = \tilde{G}^s_{l,m} \cap G^q_l, q \neq p \]
Ghost cell setting

Local synchronization

\[ \tilde{S}_{l,m}^{s,p} = \tilde{G}_{l,m}^{s,p} \cap G_{l}^{p} \]

Parallel synchronization

\[ \tilde{S}_{l,m}^{s,q} = \tilde{G}_{l,m}^{s,p} \cap G_{l}^{q}, q \neq p \]

Interpolation and physical boundary conditions remain strictly local

- Scheme \( H(\Delta t_l) \)
  evaluated locally

- Restriction and prolongation local

Ghost cell values:

- Interpolation
- Local synchronization
- Parallel synchronization
- Physical boundary
Parallel flux correction

\[ \text{Node } p \quad \text{Node } q \]

\[ \begin{align*}
\text{Structured adaptive mesh refinement}
\end{align*} \]
Parallel flux correction

1. Strictly local: Init $\delta F_{n,l+1}$ with $F^n(\tilde{G}_{l,m} \cap \partial G_{l+1}, t)$
Parallel flux correction

1. Strictly local: Init $\delta F_{n,l+1}$ with $F_n(\bar{G}_l,m \cap \partial G_{l+1}, t)$

Structured adaptive mesh refinement

Node p

Node q
Parallel flux correction

1. Strictly local: Init $\delta F^{n,l+1}$ with $F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t)$
2. Strictly local: Add $F^n(\partial G_{l,m}, t)$ to $\delta F^{n,l}$

![PointCloud](image.png)
Parallel flux correction

1. Strictly local: Init $\delta F^{n,l+1}$ with $F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t)$
2. Strictly local: Add $F^n(\partial G_{l,m}, t)$ to $\delta F^{n,l}$
3. Parallel communication: Correct $Q^l(t + \Delta t_l)$ with $\delta F^{l+1}$
The recursive algorithm in parallel

AdvanceLevel(/)

Repeat $r_l$ times
  Set ghost cells of $Q^l(t)$
  If time to regrid?
    Regrid(/)
  UpdateLevel(/)
  If level $l+1$ exists?
    Set ghost cells of $Q^l(t + \Delta t_l)$
    AdvanceLevel(/ + 1)
    Average $Q^{l+1}(t + \Delta t_l)$ onto $Q^l(t + \Delta t_l)$
    Correct $Q^l(t + \Delta t_l)$ with $\delta F^{l+1}$
  $t := t + \Delta t_l$

UpdateLevel(/)

For all $m = 1$ To $M_l$ Do
  $Q(G^s_{l,m}, t) \xrightarrow{\mathcal{H}(\Delta t_l)} Q(G_{l,m}, t + \Delta t_l), F^n(\bar{G}_{l,m}, t)$
  If level $l > 0$
    Add $F^n(\partial G_{l,m}, t)$ to $\delta F_{n,l}$
  If level $l + 1$ exists
    Init $\delta F_{n,l+1}$ with $F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t)$
The recursive algorithm in parallel

**AdvanceLevel(l)**

Repeat \( r_l \) times
- Set ghost cells of \( Q^l(t) \)
- If time to regrid?
  - Regrid(l)
- UpdateLevel(l)
- If level \( l + 1 \) exists?
  - Set ghost cells of \( Q^l(t + \Delta t_l) \)
  - AdvanceLevel(l + 1)
  - Average \( Q^{l+1}(t + \Delta t_l) \) onto \( Q^l(t + \Delta t_l) \)
  - Correct \( Q^l(t + \Delta t_l) \) with \( \delta F^{l+1} \)

\[ t := t + \Delta t_l \]

**UpdateLevel(l)**

For all \( m = 1 \) To \( M_l \) Do

\[ Q(G^s_{l,m}, t) \xrightarrow{H(\Delta t_l)} Q(G_{l,m}, t + \Delta t_l), F^n(\bar{G}_{l,m}, t) \]

If level \( l > 0 \)
- Add \( F^n(\partial G_{l,m}, t) \) to \( \delta F^{n,l} \)

If level \( l + 1 \) exists
- Init \( \delta F^{n,l+1} \) with \( F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t) \)

- Numerical update
- strictly local
- Numerical update
- strictly local
- Inter-level transfer local
- Parallel synchronization

Structured adaptive mesh refinement
The recursive algorithm in parallel

AdvanceLevel(/)

\begin{align*}
&\text{Repeat } r_l \text{ times } \\
&\quad \text{Set ghost cells of } Q^l(t) \\
&\quad \text{If time to regrid?} \\
&\quad \quad \text{Regrid(/)} \\
&\quad \text{UpdateLevel(/)} \\
&\quad \text{If level } l + 1 \text{ exists?} \\
&\quad \quad \text{Set ghost cells of } Q^l(t + \Delta t_l) \\
&\quad \quad \text{AdvanceLevel(} l + 1 \text{)} \\
&\quad \quad \text{Average } Q^{l+1}(t + \Delta t_l) \text{ onto } Q^l(t + \Delta t_l) \\
&\quad \quad \text{Correct } Q^l(t + \Delta t_l) \text{ with } \delta F^{l+1} \\
&\quad t := t + \Delta t_l
\end{align*}

UpdateLevel(/)

\begin{align*}
&\text{For all } m = 1 \text{ To } M_l \text{ Do } \\
&\quad Q(G^s_{l,m}, t) \xrightarrow{\mathcal{H}(\Delta t_l)} Q(G_{l,m}, t + \Delta t_l) , F^n(\bar{G}_{l,m}, t) \\
&\quad \text{If level } l > 0 \\
&\quad \quad \text{Add } F^n(\partial G_{l,m}, t) \text{ to } \delta F^{n,l} \\
&\quad \text{If level } l + 1 \text{ exists} \\
&\quad \quad \text{Init } \delta F^{n,l+1} \text{ with } F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t)
\end{align*}

▶ Numerical update strictly local

▶ Inter-level transfer local
The recursive algorithm in parallel

AdvanceLevel(/)

Repeat \( r_l \) times

Set ghost cells of \( Q^l(t) \)
If time to regrid?
    Regrid(/)
UpdateLevel(/)
If level \( l+1 \) exists?
    Set ghost cells of \( Q^{l+1}(t + \Delta t_l) \)
    AdvanceLevel(/ + 1)
    Average \( Q^{l+1}(t + \Delta t_l) \) onto \( Q^l(t + \Delta t_l) \)
    Correct \( Q^l(t + \Delta t_l) \) with \( \delta F^{l+1} \)
\( t := t + \Delta t_l \)

UpdateLevel(/)

For all \( m = 1 \) To \( M_l \) Do

\( Q(G^s_{l,m}, t) \xrightarrow{H(\Delta t_l)} Q(G_{l,m}, t + \Delta t_l), F^n(\bar{G}_{l,m}, t) \)
If level \( l > 0 \)
    Add \( F^n(\partial G_{l,m}, t) \) to \( \delta F^{n,l} \)
If level \( l + 1 \) exists
    Init \( \delta F^{n,l+1} \) with \( F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t) \)

▶ Numerical update strictly local
▶ Inter-level transfer local
▶ Parallel synchronization
The recursive algorithm in parallel

\textbf{AdvanceLevel}(l)

Repeat $r_l$ times

\begin{itemize}
  \item Set ghost cells of $Q^l(t)$
  \item If time to regrid?
    \begin{itemize}
      \item Regrid(l)
    \end{itemize}
  \item UpdateLevel(l)
  \item If level $l+1$ exists?
    \begin{itemize}
      \item Set ghost cells of $Q^l(t + \Delta t_l)$
      \item AdvanceLevel($l+1$)
      \item Average $Q^{l+1}(t + \Delta t_l)$ onto $Q^l(t + \Delta t_l)$
      \item Correct $Q^l(t + \Delta t_l)$ with $\delta F^{l+1}$
      \item $t := t + \Delta t_l$
    \end{itemize}
\end{itemize}

UpdateLevel(l)

For all $m = 1$ To $M_l$ Do

\begin{itemize}
  \item $Q(G^s_{l,m}, t) \xrightarrow{H(\Delta t_l)} Q(G_{l,m}, t + \Delta t_l), F^n(\bar{G}_{l,m}, t)$
  \item If level $l > 0$
    \begin{itemize}
      \item Add $F^n(\partial G_{l,m}, t)$ to $\delta F^{n,l}$
    \end{itemize}
  \item If level $l + 1$ exists
    \begin{itemize}
      \item Init $\delta F^{n,l+1}$ with $F^n(\bar{G}_{l,m} \cap \partial G_{l+1}, t)$
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item Numerical update strictly local
  \item Inter-level transfer local
  \item Parallel synchronization
  \item Application of $\delta F^{l+1}$ on $\partial G^q_l$
\end{itemize}
The recursive algorithm in parallel

\begin{align*}
\text{AdvanceLevel}(l) & \\
\text{Repeat } r_l \text{ times} & \\
\text{Set ghost cells of } Q^l(t) & \\
\text{If time to regrid?} & \\
\text{Regrid}(l) & \\
\text{UpdateLevel}(l) & \\
\text{If level } l + 1 \text{ exists?} & \\
\text{Set ghost cells of } Q^l(t + \Delta t_l) & \\
\text{AdvanceLevel}(l + 1) & \\
\text{Average } Q^{l+1}(t + \Delta t_l) \text{ onto } Q^l(t + \Delta t_l) & \\
\text{Correct } Q^l(t + \Delta t_l) \text{ with } \delta F^{l+1} & \\
t := t + \Delta t_l & \\
\text{UpdateLevel}(l) & \\
\text{For all } m = 1 \text{ To } M_l \text{ Do} & \\
Q(G^s_l,m,t) \xrightarrow{\mathcal{H}(\Delta t_l)} Q(G_{l,m},t + \Delta t_l), F^n(\bar{G}_{l,m},t) & \\
\text{If level } l > 0 & \\
\text{Add } F^n(\partial G_{l,m},t) \text{ to } \delta F^{n,l} & \\
\text{If level } l + 1 \text{ exists} & \\
\text{Init } \delta F^{n,l+1} \text{ with } F^n(\bar{G}_{l,m} \cap \partial G_{l+1},t) & \\
\end{align*}

- Numerical update strictly local
- Inter-level transfer local
- Parallel synchronization
- Application of $\delta F^{l+1}$ on $\partial G^q_l$
## Regridding algorithm in parallel

Regrid($l$) - Regrid all levels $l > l$

For $l = l_f$ Down to $l$ Do

1. Flag $N^l$ according to $Q^l(t)$
2. If level $l + 1$ exists?
   - Flag $N^l$ below $\tilde{G}^{l+2}$
   - Flag buffer zone on $N^l$
   - Generate $\tilde{G}^{l+1}$ from $N^l$

$\tilde{G}_l := G_l$

For $l = l$ To $l_f$ Do

1. $C \tilde{G}_l := G_0 \setminus \tilde{G}_l$
2. $\tilde{G}_{l+1} := \tilde{G}_{l+1} \setminus C \tilde{G}_l$

Recompose($l$)
Regridding algorithm in parallel

Regrid(l) - Regrid all levels \( l > l \)

For \( l = l_f \) Down to \( l \) Do

- Flag \( N^l \) according to \( Q^l(t) \)
- If level \( l + 1 \) exists?
  - Flag \( N^l \) below \( \tilde{G}^{l+2} \)
- Flag buffer zone on \( N^l \)
- Generate \( \tilde{G}^{l+1} \) from \( N^l \)

\( \tilde{G}_l := G_l \)

For \( l = l \) To \( l_f \) Do

- \( C\tilde{G}_l := G_0 \setminus \tilde{G}_l \)
- \( \tilde{G}_{l+1} := \tilde{G}_{l+1} \setminus C\tilde{G}_l \)

Recompose(l)
A parallel SAMR algorithm

Regridding algorithm in parallel

Regrid(ι) - Regrid all levels ι > l

For ι = ι_f Do downto ι_l Do
  Flag N_ι according to Q_ι(t)
  If level ι + 1 exists?
    Flag N_ι below G_ι+2
  Flag buffer zone on N_ι
  Generate G_ι+1 from N_ι

G_ι := G_ι

For ι = ι_l To ι_f Do
  C G_ι := G_0 \ G_ι
  G_ι+1 := G_ι+1 \ C G_ι

Recompose(ι)

▶ Need a ghost cell overlap of b cells to ensure correct setting of refinement flags in parallel
Regridding algorithm in parallel

Regrid(\(l\)) - Regrid all levels \(\iota > l\)

For \(\iota = l_f\) Down to \(l\) Do

Flag \(N^l\) according to \(Q^\iota(t)\)
If level \(\iota + 1\) exists?

Flag \(N^l\) below \(\tilde{G}^{\iota+2}\)

Flag buffer zone on \(N^l\)
Generate \(\tilde{G}^{\iota+1}\) from \(N^l\)

\(\tilde{G}_l := G_l\)

For \(\iota = l\) To \(l_f\) Do

\(C\tilde{G}_l := G_0 \backslash \tilde{G}_l\)
\(\tilde{G}_{l+1} := \tilde{G}_{l+1} \backslash C\tilde{G}_l\)

Recompose(\(l\))

- Need a ghost cell overlap of \(b\) cells to ensure correct setting of refinement flags in parallel
- Two options exist (we choose the latter):
  - Global clustering algorithm
  - Local clustering algorithm and concatenation of new lists \(\tilde{G}^{\iota+1}\)
Regridding algorithm in parallel

Regrid($l$) – Regrid all levels $l > l$

For $i = l_f$ Down to $l$ Do
   Flag $N^i$ according to $Q^i(t)$
   If level $i+1$ exists?
      Flag $N^i$ below $G^{i+2}$
      Flag buffer zone on $N^i$
   Generate $G^{i+1}$ from $N^i$

$G_l := G_l$
For $i = l$ To $l_f$ Do
   $C \tilde{G}_i := G_0 \setminus \tilde{G}_l$
   $\tilde{G}_{i+1} := \tilde{G}_{i+1} \setminus C \tilde{G}_l$

Recompose($l$)

- Need a ghost cell overlap of $b$ cells to ensure correct setting of refinement flags in parallel
- Two options exist (we choose the latter):
  - Global clustering algorithm
  - Local clustering algorithm and concatenation of new lists $G^{i+1}$
Regridding algorithm in parallel

Regrid($l$) - Regrid all levels $\iota > l$

For $\iota = l_f$ Downto $l$ Do
  Flag $N^\iota$ according to $Q^\iota(t)$
  If level $\iota + 1$ exists?
    Flag $N^\iota$ below $\tilde{G}^{\iota+2}$
    Flag buffer zone on $N^\iota$
    Generate $\tilde{G}^{\iota+1}$ from $N^\iota$

$\tilde{G}_l := G_l$

For $\iota = l$ To $l_f$ Do
  $C\tilde{G}_\iota := G_0 \setminus \tilde{G}_\iota$
  $\tilde{G}_{\iota+1} := \tilde{G}_{\iota+1} \setminus C\tilde{G}_\iota$
  $\tilde{G}_{\iota+1} := \tilde{G}_{\iota+1} \setminus C\tilde{G}_\iota$

Recompose($l$)

- Need a ghost cell overlap of $b$ cells to ensure correct setting of refinement flags in parallel
- Two options exist (we choose the latter):
  - Global clustering algorithm
  - Local clustering algorithm and concatenation of new lists $\tilde{G}^{\iota+1}$
Recomposition algorithm in parallel

Recompose(l) - Reorganize all levels

For $\iota = l + 1$ To $l_f + 1$ Do

Interpolate $Q^{\iota-1}(t)$ onto $\tilde{Q}^{\iota}(t)$

Copy $Q^{\iota}(t)$ onto $\tilde{Q}^{\iota}(t)$
Set ghost cells of $\tilde{Q}^{\iota}(t)$
$Q^{\iota}(t) := \tilde{Q}^{\iota}(t)$
$G_\iota := \tilde{G}_\iota$
Recomposition algorithm in parallel

Recompose(l) - Reorganize all levels

Generate \( G_p^0 \) from \( \{G_0, \ldots, G_l, \tilde{G}_{l+1}, \ldots, \tilde{G}_{l_f+1}\} \)

For \( \iota = 0 \) To \( l_f + 1 \) Do

Interpolate \( Q^{\iota-1}(t) \) onto \( \tilde{Q}^{\iota}(t) \)

Copy \( Q^{\iota}(t) \) onto \( \tilde{Q}^{\iota}(t) \)

Set ghost cells of \( \tilde{Q}^{\iota}(t) \)

\( Q^{\iota}(t) := \tilde{Q}^{\iota}(t) \)

\( G^p_{\iota} := \tilde{G}^p_{\iota}, \ G_{\iota} := \bigcup_p G^p_{\iota} \)

\( \triangleright \) Global redistribution can also be required when regridding higher levels and \( G_0, \ldots, G_l \) do not change (drawback of domain decomposition)
Recomposition algorithm in parallel

Recompose(\(l\)) - Reorganize all levels

Generate \(G_0^p\) from \(\{G_0, ..., G_l, \tilde{G}_{l+1}, ..., \tilde{G}_{l+1}\}\)

For \(\iota = 0\) To \(l_f + 1\) Do

If \(\iota > l\)

\[\tilde{G}_\iota^p := \tilde{G}_\iota \cap G_0^p\]

Interpolate \(Q^{\iota-1}(t)\) onto \(\tilde{Q}^\iota(t)\)

Global redistribution can also be required when regridding higher levels and \(G_0, ..., G_l\) do not change (drawback of domain decomposition)

When \(\iota > l\) do nothing special

For \(\iota \leq l\), redistribute additionally

Copy \(Q^\iota(t)\) onto \(\tilde{Q}^\iota(t)\)

Set ghost cells of \(\tilde{Q}^\iota(t)\)

\(Q^\iota(t) := \tilde{Q}^\iota(t)\)

\(G_i^p := \tilde{G}_i^p, \quad G_i := \bigcup_p G_i^p\)
Recomposition algorithm in parallel

Recompose(l) - Reorganize all levels

Generate $G^p_0$ from $\{G_0, ..., G_l, \bar{G}_{l+1}, ..., \bar{G}_{l_f+1}\}$

For $\iota = 0$ To $l_f + 1$ Do
  If $\iota > l$
    $\bar{G}^p_{\iota} := \bar{G}_{\iota} \cap G^p_0$
    Interpolate $Q^{\iota-1}(t)$ onto $\bar{Q}^\iota(t)$
  else
    $\bar{G}^p_{\iota} := G_{\iota} \cap G^p_0$
    If $\iota > 0$
      Copy $\delta F^{n,\iota}$ onto $\delta \tilde{F}^{n,\iota}$
      $\delta F^{n,\iota} := \delta \tilde{F}^{n,\iota}$
  Copy $Q^\iota(t)$ onto $\bar{Q}^\iota(t)$
  Set ghost cells of $\bar{Q}^\iota(t)$
  $Q^\iota(t) := \bar{Q}^\iota(t)$
  $G^p_{\iota} := \bar{G}^p_{\iota}$, $G_{\iota} := \bigcup_p G^p_{\iota}$

- Global redistribution can also be required when regridding higher levels and $G_0, ..., G_l$ do not change (drawback of domain decomposition)
- When $\iota > l$ do nothing special
- For $\iota \leq l$, redistribute additionally
  - Flux corrections $\delta F^{n,\iota}$
Recomposition algorithm in parallel

Recompose(\(l\)) - Reorganize all levels

Generate \(G^P_0\) from \(\{G_0, ..., G_l, \check{G}_{l+1}, ..., \check{G}_{l_f+1}\}\)

For \(\iota = 0\) To \(l_f + 1\) Do

If \(\iota > l\)

\(\check{G}^P_\iota := \check{G}_\iota \cap G^P_0\)

Interpolate \(Q^{\iota-1}(t)\) onto \(\check{Q}^\iota(t)\)

else

\(\check{G}^P_\iota := G_\iota \cap G^P_0\)

If \(\iota > 0\)

Copy \(\delta F^{n,\iota}\) onto \(\check{\delta F}^{n,\iota}\)

\(\delta F^{n,\iota} := \check{\delta F}^{n,\iota}\)

If \(\iota \geq l\) then \(\kappa_\iota = 0\) else \(\kappa_\iota = 1\)

For \(\kappa = 0\) To \(\kappa_\iota\) Do

Copy \(Q^\iota(t + \kappa \Delta t_\iota)\) onto \(\check{Q}^\iota(t + \kappa \Delta t_\iota)\)

Set ghost cells of \(\check{Q}^\iota(t + \kappa \Delta t_\iota)\)

\(Q^\iota(t + \kappa \Delta t_\iota) := \check{Q}^\iota(t + \kappa \Delta t_\iota)\)

\(G^P_\iota := \check{G}^P_\iota, \ G_\iota := \bigcup_P G^P_\iota\)

- Global redistribution can also be required when regridding higher levels and \(G_0, ..., G_l\) do not change (drawback of domain decomposition)
- When \(\iota > l\) do nothing special
- For \(\iota \leq l\), redistribute additionally
  - Flux corrections \(\delta F^{n,\iota}\)
  - Already updated time level \(Q^\iota(t + \kappa \Delta t_\iota)\)
Recomposition algorithm in parallel

Recompose(\(l\)) - Reorganize all levels

Generate \(G^0_p\) from \(\{G_0, \ldots, G_l, \tilde{G}_{l+1}, \ldots, \tilde{G}_{l_f+1}\}\)

For \(\iota = 0\) To \(l_f + 1\) Do

If \(\iota > l\)

\(\tilde{G}^p_\iota := \tilde{G}_\iota \cap G^0_p\)

Interpolate \(Q^{\iota-1}(t)\) onto \(\tilde{Q}^\iota(t)\)

else

\(\tilde{G}^p_\iota := G_\iota \cap G^0_p\)

If \(\iota > 0\)

Copy \(\delta F^{n,\iota}\) onto \(\delta \tilde{F}^{n,\iota}\)

\(\delta F^{n,\iota} := \delta \tilde{F}^{n,\iota}\)

If \(\iota \geq l\) then \(\kappa_\iota = 0\) else \(\kappa_\iota = 1\)

For \(\kappa = 0\) To \(\kappa_\iota\) Do

Copy \(Q^{\iota}(t + \kappa \Delta t_\iota)\) onto \(\tilde{Q}^{\iota}(t + \kappa \Delta t_\iota)\)

Set ghost cells of \(\tilde{Q}^{\iota}(t + \kappa \Delta t_\iota)\)

\(Q^{\iota}(t + \kappa \Delta t_\iota) := \tilde{Q}^{\iota}(t + \kappa \Delta t_\iota)\)

\(G^p_\iota := \tilde{G}^p_\iota, G_\iota := \bigcup_p G^p_\iota\)

- Global redistribution can also be required when regridding higher levels and \(G_0, \ldots, G_l\) do not change (drawback of domain decomposition)

- When \(\iota > l\) do nothing special

- For \(\iota \leq l\), redistribute additionally

  - Flux corrections \(\delta F^{n,\iota}\)

  - Already updated time level \(Q^{\iota}(t + \kappa \Delta t_\iota)\)
Space-filling curve algorithm

Calculation domain

- High Workload
- Medium Workload
- Low Workload
Space-filling curve algorithm

- High Workload
- Medium Workload
- Low Workload
Space-filling curve algorithm

Calculation domain

Necessary domain of Space-Filling Curve

High Workload
Medium Workload
Low Workload
Space-filling curve algorithm

Calculation domain

Necessary domain of Space-Filling Curve

High Workload
Medium Workload
Low Workload
Space-filling curve algorithm

Calculation domain

Necessary domain of Space-Filling Curve

Proc. 1

- High Workload
- Medium Workload
- Low Workload

Structured adaptive mesh refinement
Space-filling curve algorithm

Calculation domain

Necessary domain of Space-Filling Curve

- Yellow: Proc. 1 (High Workload)
- Blue: Proc. 2 (Medium Workload)
- Other: Low Workload
Space-filling curve algorithm

Calculation domain

Necessary domain of Space-Filling Curve

<table>
<thead>
<tr>
<th>Proc. 1</th>
<th>High Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc. 2</td>
<td>Medium Workload</td>
</tr>
<tr>
<td>Proc. 3</td>
<td>Low Workload</td>
</tr>
</tbody>
</table>
Overview

“Adaptive Mesh Refinement in Object-oriented C++”

~46,000 LOC for C++ SAMR kernel, ~140,000 total C++, C, Fortran-77

uses parallel hierarchical data structures that have evolved from DAGH

Implements explicit SAMR with different finite volume solvers

Embedded boundary method, FSI coupling

The Virtual Test Facility: AMROC V2.0 plus solid mechanics solvers

~430,000 lines of code total in C++, C, Fortran-77, Fortran-90

autoconf / automake environment with support for typical parallel high-performance system

http://www.vtf.website [Deiterding et al., 2006][Deiterding et al., 2007]
UML design of AMROC

- Classical framework approach with generic main program in C++
- Classical framework approach with generic main program in C++
- Customization / modification in Problem.h include file by derivation from base classes and redefining virtual interface functions
UML design of AMROC

- Classical framework approach with generic main program in C++
- Customization / modification in Problem.h include file by derivation from base classes and redefining virtual interface functions
- Predefined, scheme-specific classes provided for standard simulations
UML design of AMROC

- Classical framework approach with generic main program in C++
- Customization / modification in Problem.h include file by derivation from base classes and redefining virtual interface functions
- Predefined, scheme-specific classes provided for standard simulations
- Clawpack, WENO: Standard simulations require only linking to F77 functions for initial and boundary conditions, source terms. No C++ knowledge required
UML design of AMROC

- Classical framework approach with generic main program in C++
- Customization / modification in Problem.h include file by derivation from base classes and redefining virtual interface functions
- Predefined, scheme-specific classes provided for standard simulations
- Clawpack, WENO: Standard simulations require only linking to F77 functions for initial and boundary conditions, source terms. No C++ knowledge required
- Expert usage (algorithm modification, advanced output, etc.) in C++
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \approx l_{\text{max}} \prod_{\kappa=l+1}^{l_{\text{max}}} r_\kappa$ to uniquely identify a cell within the hierarchy
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \approx \prod_{\kappa=l+1}^{l_{\text{max}}} r_\kappa$ to uniquely identify a cell within the hierarchy

- Box<dim>, BoxList<dim> class that define rectangular regions $G_{m,l}$ by lowerleft, upperright, stepsize and specify topological operations $\cap$, $\cup$, $\setminus$
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \approx \prod_{\kappa=l+1}^{l_{\text{max}}} r_\kappa$ to uniquely identify a cell within the hierarchy.

- `Box<dim>, BoxList<dim>` class that define rectangular regions $G_{m,l}$ by lowerleft, upperright, stepsize and specify topological operations $\cap, \cup, \setminus$.

- `Patch<dim,type>` class that assigns data to a rectangular grid $G_{m,l}$.
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \simeq \prod_{\kappa=l+1}^{l_{\text{max}}} r_{\kappa}$ to uniquely identify a cell within the hierarchy

- Box<dim>, BoxList<dim> class that define rectangular regions $G_{m,l}$ by lowerleft, upperright, stepsize and specify topological operations $\cap, \cup, \setminus$

- Patch<dim,type> class that assigns data to a rectangular grid $G_{m,l}$

- A class, here GridFunction<dim,type>, that defines topological relations between lists of Patch objects to implement synchronization, restriction, prolongation, re-distribution
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \cong l_{\text{max}} \prod_{\kappa=l+1} r_\kappa$ to uniquely identify a cell within the hierarchy.
- `Box<dim>`, `BoxList<dim>` class that define rectangular regions $G_{m,l}$ by `lowerleft`, `upperright`, `stepsize` and specify topological operations $\cap$, $\cup$, $\setminus$.
- `Patch<dim,type>` class that assigns data to a rectangular grid $G_{m,l}$.
- A class, here `GridFunction<dim,type>`, that defines topological relations between lists of `Patch` objects to implement synchronization, restriction, prolongation, re-distribution.
- Hierarchical parallel data structures are typically C++, routines on patches often Fortran.
Hierarchical data structures

Directory amroc/hds. Key classes:

- **Coords**: Point in index coordinator system
  
  [code](amroc/doc/html/hds/classCoords.html)

- **BBox**: Rectangular region
  
  [code](amroc/doc/html/hds/classBBox.html)

- **BBoxList**: Set of BBox elements
  
  [code](amroc/doc/html/hds/classBBoxList.html)

- **GridBox**: Has a BBox member, but adds level and partitioning information
  
  [code](amroc/doc/html/hds/classGridBox.html)

- **GridBoxList**: Set of GridBox elements
  
  [code](amroc/doc/html/hds/classBBoxList.html)

- **GridData**: Creates array data of Type of same dimension as BBox, has extensive math operators
  
  [code](amroc/doc/html/hds/classGridData_3_01Type_00_012_01_4.html)

- **Vector**: Vector of state is usually Vector<double, N>
  
  [code](amroc/doc/html/hds/classVector.html)
Hierarchical data structures - II

- **GridDataBlock**<sub>Type, dim</sub>: The Patch-class. Has a GridData<sub>Type, dim</sub> member, knows about relations of current patch within AMR hierarchy

  code/amroc/doc/html/hds/classGridDataBlock.html

- **GridFunction**<sub>Type, dim</sub>: Uses GridDataBlock<sub>Type, dim</sub> objects to organize hierarchical data of Type after receiving GridBoxLists. Has extensive math operators for whole levels. Recreates GridDataBlock<sub>Type, dim</sub> lists automatically when GridBoxList changes. Calls interlevel operations are automatically when required.

  code/amroc/doc/html/hds/classGridFunction.html

- **GridHierarchy**<sub>Type, dim</sub>: Uses sets of GridBoxList to organize topology of the hierarchy. All GridFunction<sub>Type, dim</sub> are members and receive updated GridBoxList after regridding and repartitioning. Calls DAGHDistribution of partitioning. Implements parallel Recompose().

  code/amroc/doc/html/hds/classGridHierarchy.html
AMR level

Directory amroc/amr. Central class is `AMRSolver< VectorType, FixupType, FlagType, dim >`:

code/amroc/doc/html/amr/classAMRSolver.html

- Uses `Integrator< VectorType, dim >` to interface and call the patch-wise numerical update

code/amroc/doc/html/amr/classIntegrator.html

- Uses `InitialCondition< VectorType, dim >` to call initial conditions patch-wise

code/amroc/doc/html/amr/classInitialCondition.html

- Uses `BoundaryConditions< VectorType, dim >` to call boundary conditions per side and patch

code/amroc/doc/html/amr/classBoundaryConditions.html

- Fortran interfaces to above classes are in amroc/amr/F77Interfaces, convenient C++ interfaces in amroc/amr/Interfaces.

- Implements parallel `AdvanceLevel()`, `RegridLevel()`.
AMR level - II

- **AMRFixup**<VectorType, FixupType, dim> implements the conservative flux correction, holds lower dimensional GridFunctions for correction terms
  
  code/amroc/doc/html/amr/classAMRFixup.html

- **AMRFlagging**<VectorType, FixupType, FlagType, dim> calls a list of refinement criteria and stores results in scalar GridFunction for flags. All criteria are in amroc/amr/Criteria
  
  code/amroc/doc/html/amr/classAMRFlagging.html

- **LevelTransfer**<VectorType, dim> provides patch-wise interpolation and restriction routines that are passed as parameters to GridFunction
  
  code/amroc/doc/html/amr/classLevelTransfer.html

- **AMRTimeStep** implements time step control for a Solver
  
  code/amroc/doc/html/amr/classAMRTimeStep.html

- **AMRInterpolation**<VectorType, dim> is an interpolation at arbitrary point location, typically used for post-processing
  
  code/amroc/doc/html/amr/classAMRInterpolation.html
References I


References II


References III


References IV
