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

Outline

Outline

Outline

Outline

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

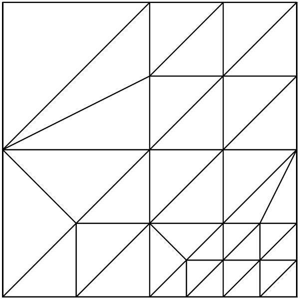
- Base grid
- Solver
- Error indicators
- Grid manipulation
- Interpolation (restriction and prolongation)

Elements of adaptive algorithms

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

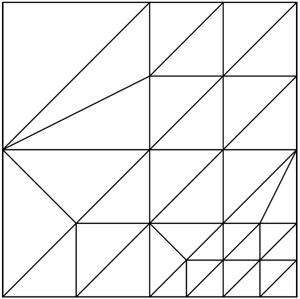
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones



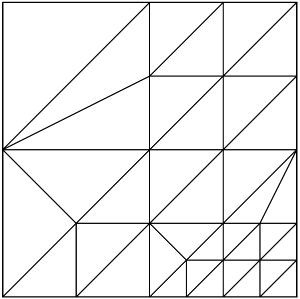
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step



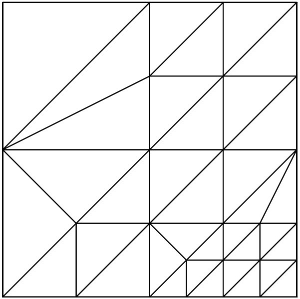
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures



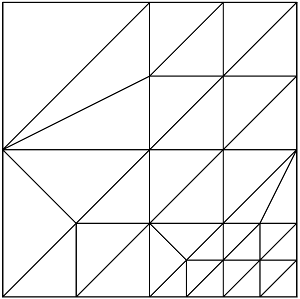
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to stored



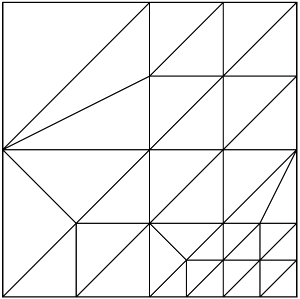
Adaptivity on unstructured meshes

- Coarse cells replaced by finer ones
- Global time-step
- Cell-based data structures
- Neighborhoods have to stored
- \+ Geometric flexible



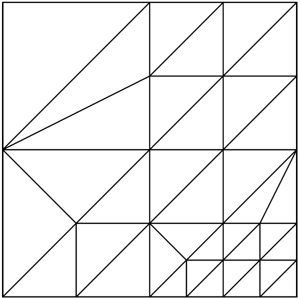
Adaptivity on unstructured meshes

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



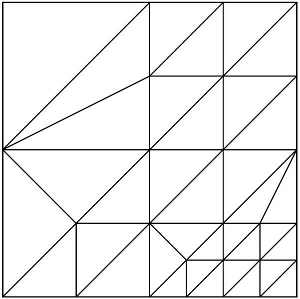
Adaptivity on unstructured meshes

- 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 stored

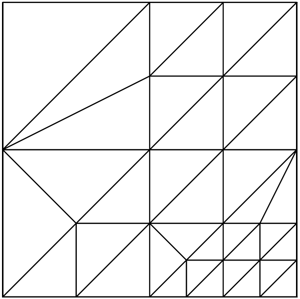
\+ Geometric flexible

\+ No hanging nodes

\+ Easy to implement

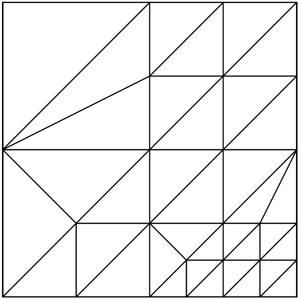
\- Higher order difficult to achieve

\- Cell aspect ratio must be considered



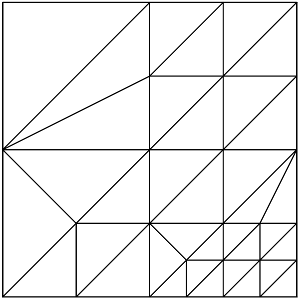
Adaptivity on unstructured meshes

- 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



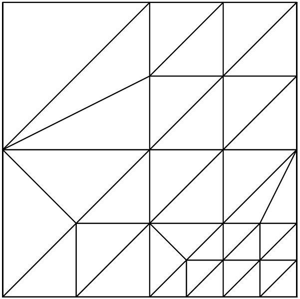
Adaptivity on unstructured meshes

- 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 / vectorizaton nearly impossible



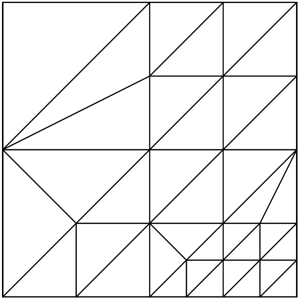
Adaptivity on unstructured meshes

- ▶ 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 / vectorizaton 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 / vectorizaton nearly impossible
- \- Complex load-balancing
- \- Complex synchronization



Structured mesh refinement techniques

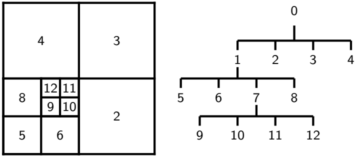
- Block-based data of equal size

Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree

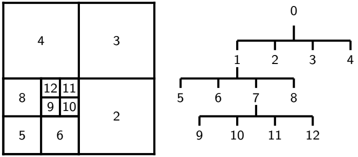
Structured mesh refinement techniques

- Block-based data of equal size
- Block stored in a quad-tree



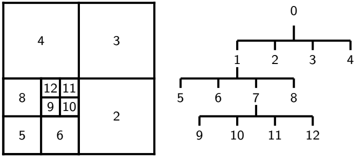
Structured mesh refinement techniques

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



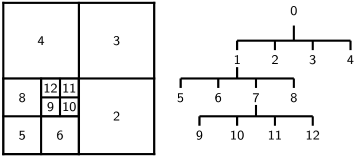
Structured mesh refinement techniques

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



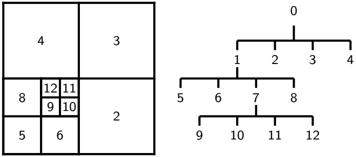
Structured mesh refinement techniques

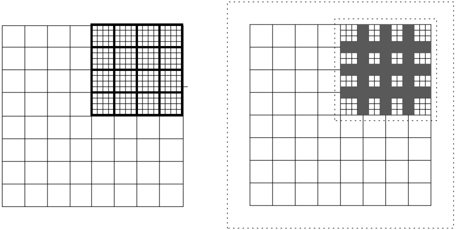
- Block-based data of equal size
- Block stored in a quad-tree
- Time-step refinement
- Global index coordinate system
- Neighborhoods need not be stored



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



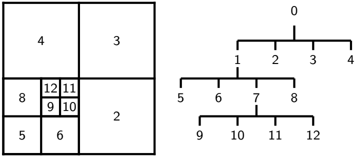


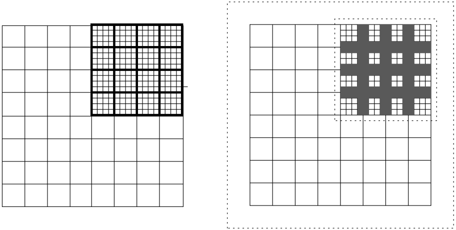
Wasted boundary space in a quad-tree

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

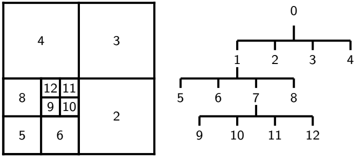


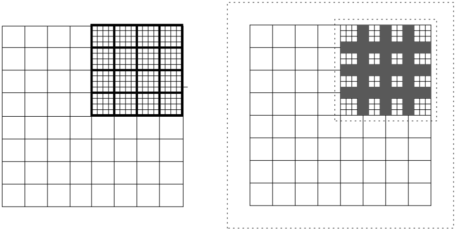


Wasted boundary space in a quad-tree

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



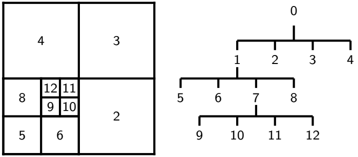


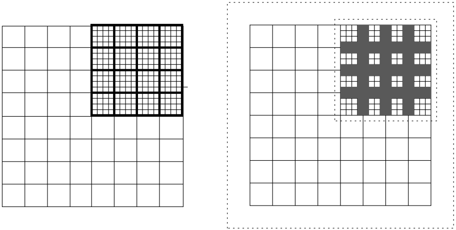
Wasted boundary space in a quad-tree

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

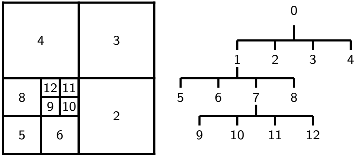


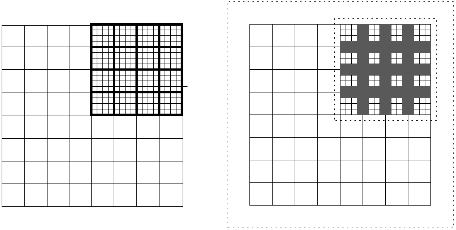


Wasted boundary space in a quad-tree

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

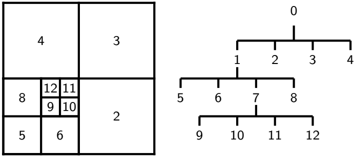
\+ Numerical scheme only for single regular block necessary

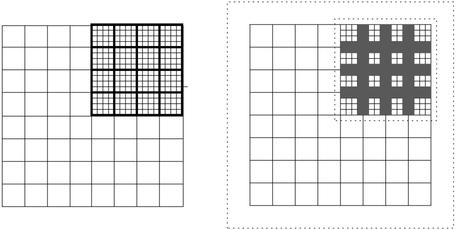
\+ Easy to implement

\+ Simple load-balancing

\+ Parent/Child relations according to tree

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

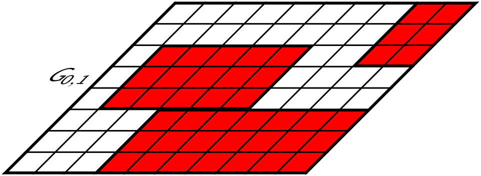




Wasted boundary space in a quad-tree

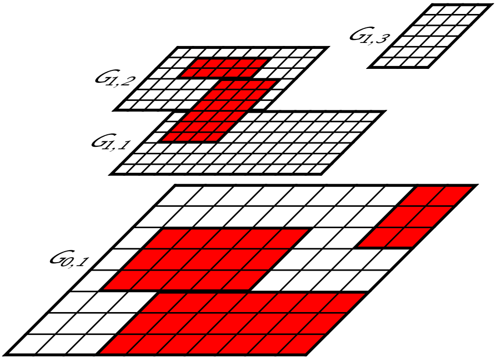
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones



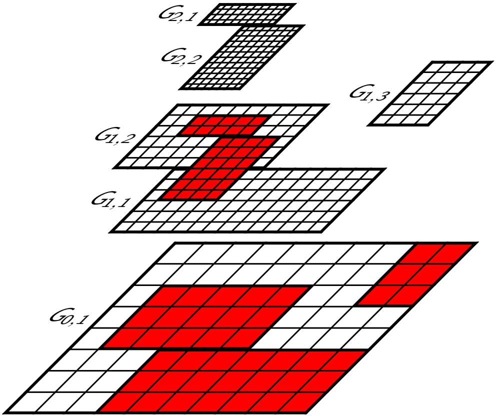
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones



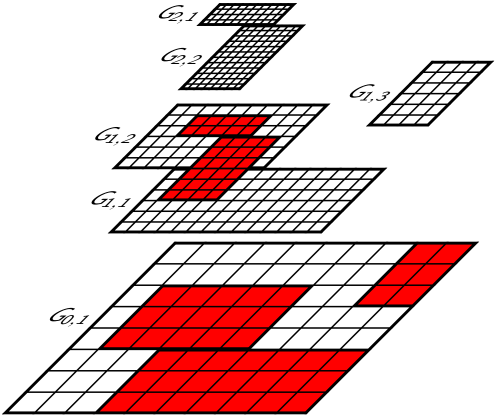
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones



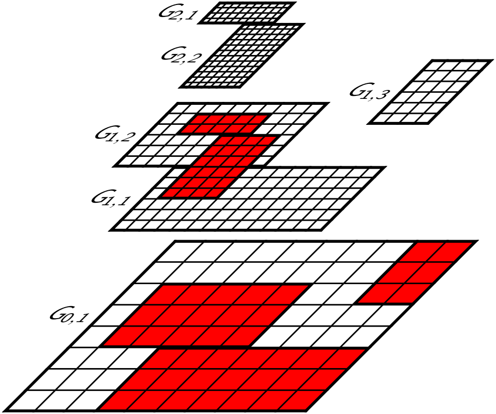
Block-structured adaptive mesh refinement (SAMR)

- Refined block overlay coarser ones
- Time-step refinement



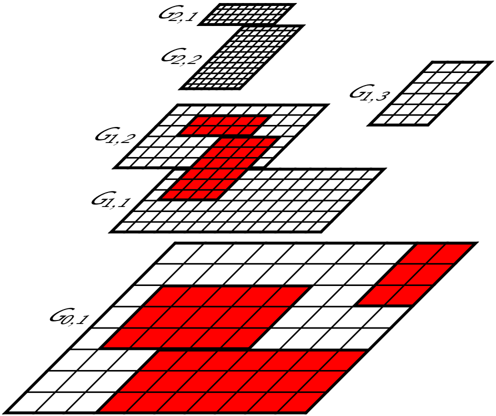
Block-structured adaptive mesh refinement (SAMR)

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



Block-structured adaptive mesh refinement (SAMR)

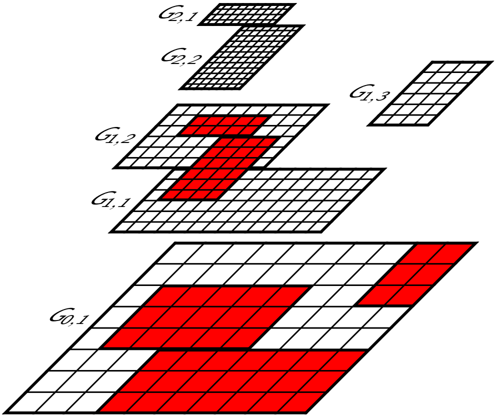
- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system



Block-structured adaptive mesh refinement (SAMR)

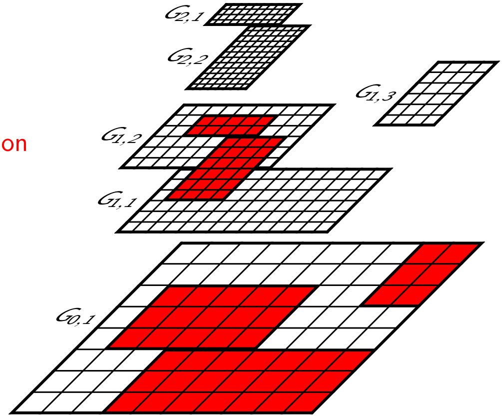
- Refined block overlay coarser ones
- Time-step refinement
- Block (aka patch) based data structures
- Global index coordinate system

\+ Numerical scheme only for single patch necessary



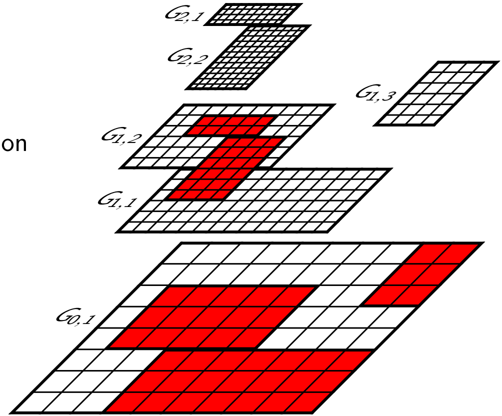
Block-structured adaptive mesh refinement (SAMR)

- 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



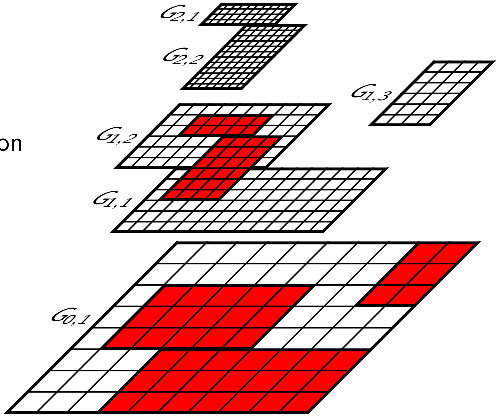
Block-structured adaptive mesh refinement (SAMR)

- 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



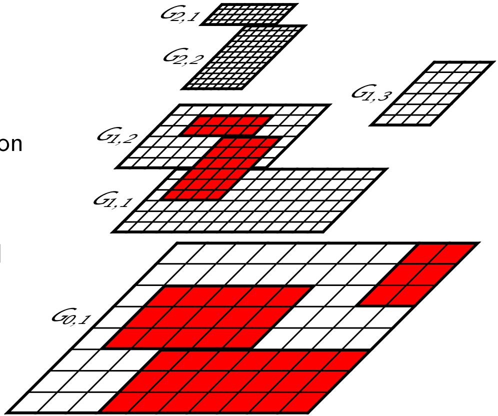
Block-structured adaptive mesh refinement (SAMR)

- 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



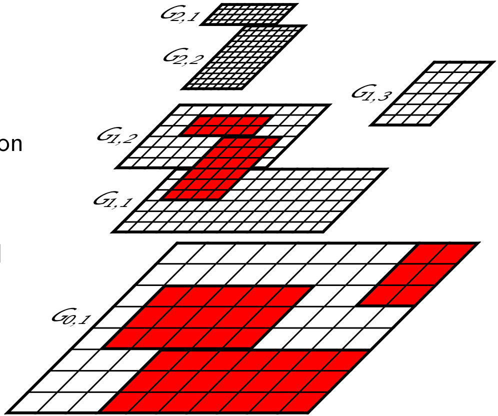
Block-structured adaptive mesh refinement (SAMR)

- 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



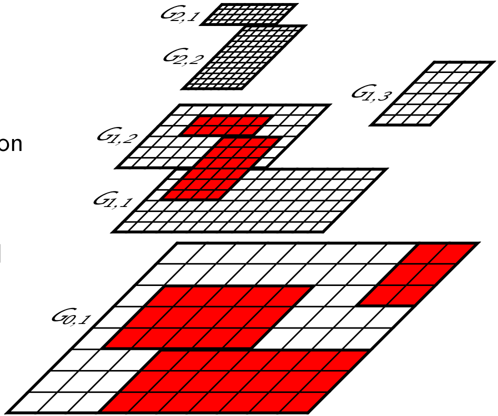
Block-structured adaptive mesh refinement (SAMR)

- 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



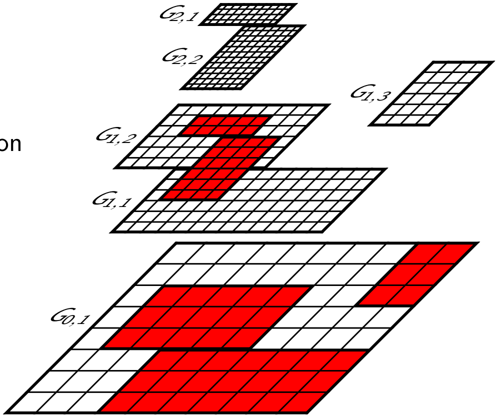
Block-structured adaptive mesh refinement (SAMR)

- ▶ 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 states.

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

- 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

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

- 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
- 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

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

- 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
- 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
- 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

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

- 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
- 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
- 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
- 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

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
  - https://computation-rnd.llnl.gov/SAMRAI/software.php

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
  - https://computation-rnd.llnl.gov/SAMRAI/software.php
- 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
  - https://ccse.lbl.gov/Downloads/index.html

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
  - https://computation-rnd.llnl.gov/SAMRAI/software.php
- 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
  - https://ccse.lbl.gov/Downloads/index.html
- 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

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
- 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

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

Further SAMR software

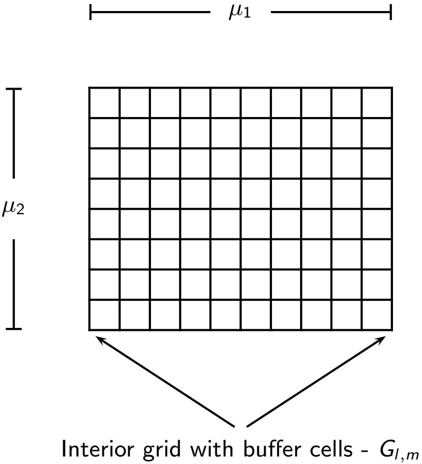
- Overture (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - http://www.overtureframework.org
- AMRClaw within Clawpack [Berger and LeVeque, 1998]
  - 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

The mth refinement grid on level l

Notations:

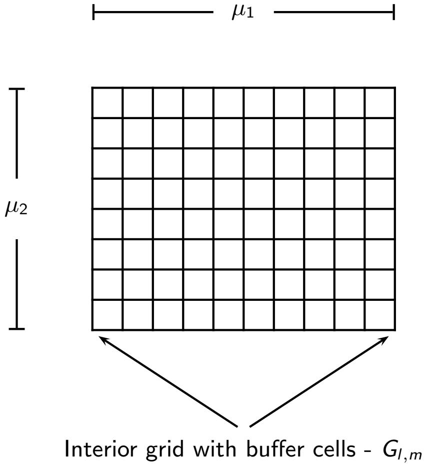
- Boundary: $\partial G_{l,m}$



The mth refinement grid on level l

Notations:

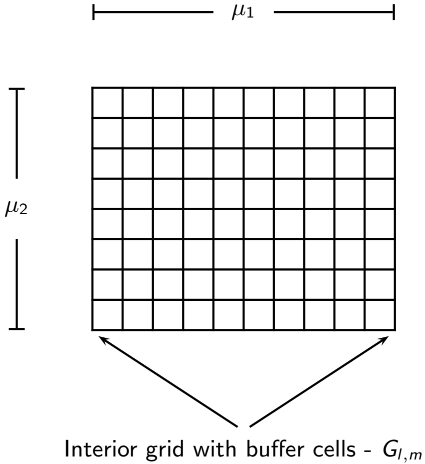
- Boundary: $\partial G_{l,m}$
- Hull: $\bar{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$



The mth refinement grid on level l

Notations:

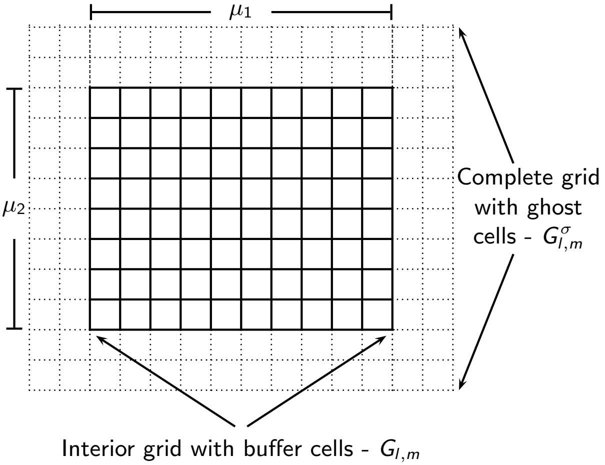
- Boundary: $\partial G_{l,m}$
- Hull:
  $\bar{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$



The mth refinement grid on level l

Notations:

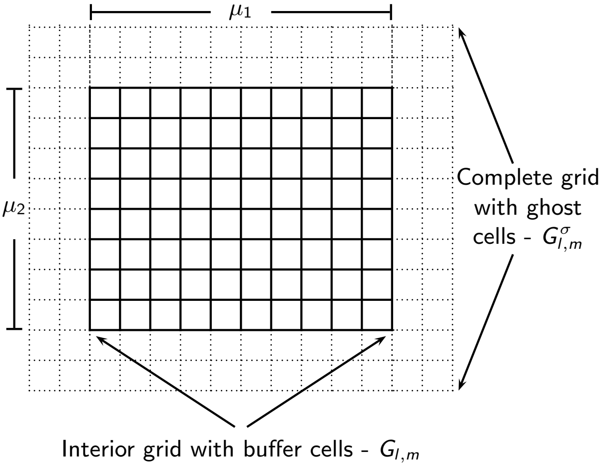
- Boundary: $\partial G_{l,m}$
- Hull: $\bar{G}_{l,m} = G_{l,m} \cup \partial G_{l,m}$



The mth 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:
  $\tilde{G}^{\sigma}_{l,m} = G^{\sigma}_{l,m} \backslash \bar{G}_{l,m}$



Refinement data

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

Refinement data

- Resolution: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- Refinement factor: $r_l \in \mathbb{N}, r_l \geq 2$ for $l > 0$ and $r_0 = 1$

Refinement data

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

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

Refinement data

- Resolution: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- 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]:
  $$\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$$
- Computational Domain: $G_0 = \bigcup_{m=1}^{M_0} G_{0,m}$

Refinement data

- Resolution: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- 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]:
$$\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$$
- 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: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- 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]:
  $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$
- 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: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- 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]:
  $$\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$$
- 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: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- 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]:

  $$\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$$
- 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: $\mathbf{Q}^l := \bigcup_m \mathbf{Q}(G_{l,m}^s)$

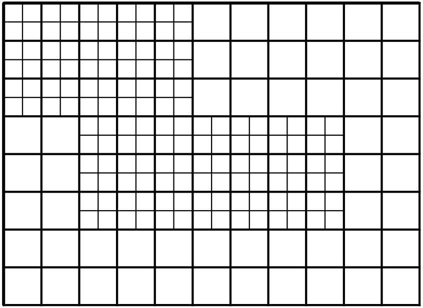
Refinement data

- Resolution: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- 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]:
$$\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$$
- 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: $\mathbf{Q}^l := \bigcup_m \mathbf{Q}(G_{l,m}^s)$
  - Numerical fluxes: $\mathbf{F}^{n,l} := \bigcup_m \mathbf{F}^n(\bar{G}_{l,m})$

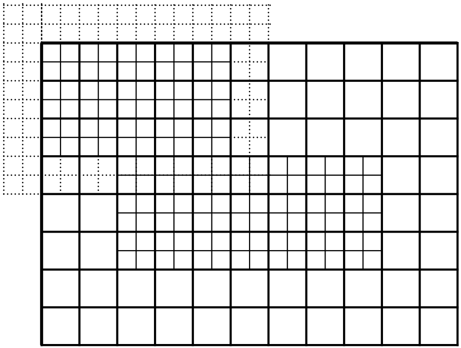
Refinement data

- Resolution: $\Delta t_l := \dfrac{\Delta t_{l-1}}{r_l}$ and $\Delta x_{n,l} := \dfrac{\Delta x_{n,l-1}}{r_l}$
- 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]:
  $$\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$$
- 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: $\mathbf{Q}^l := \bigcup_m \mathbf{Q}(G^s_{l,m})$
  - Numerical fluxes: $\mathbf{F}^{n,l} := \bigcup_m \mathbf{F}^n(\bar{G}_{l,m})$
  - Flux corrections: $\delta\mathbf{F}^{n,l} := \bigcup_m \delta\mathbf{F}^n(\partial G_{l,m})$

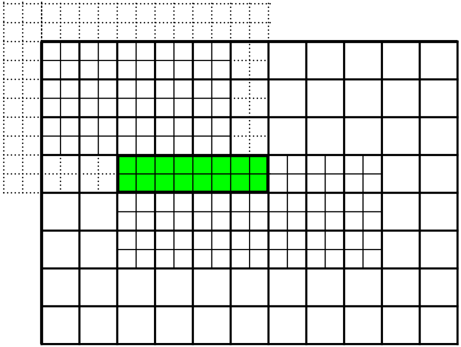
Setting of ghost cells



Setting of ghost cells

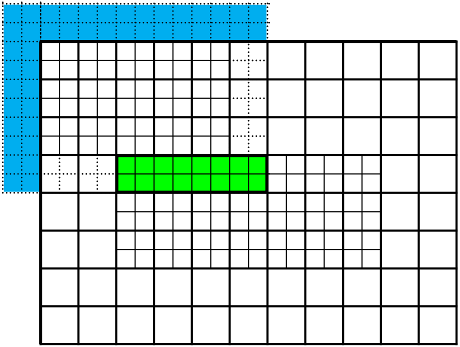


Setting of ghost cells



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

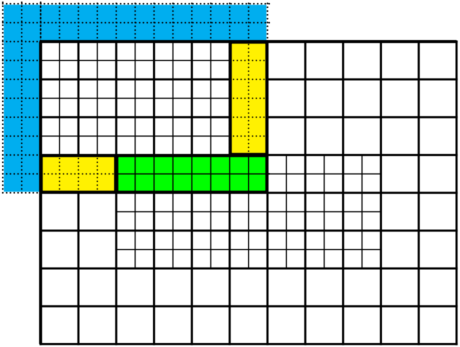
Setting of ghost cells



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

Physical boundary conditions - $\tilde{P}^s_{l,m} = \tilde{G}^s_{l,m} \backslash G_0$

Setting of ghost cells



- Synchronization with G_l - $\tilde{S}^s_{l,m} = \tilde{G}^s_{l,m} \cap G_l$
- Physical boundary conditions - $\tilde{P}^s_{l,m} = \tilde{G}^s_{l,m} \backslash G_0$
- Interpolation from G_{l-1} - $\tilde{I}^s_{l,m} = \tilde{G}^s_{l,m} \backslash (\tilde{S}^s_{l,m} \cup \tilde{P}^s_{l,m})$

Numerical update

Time-explicit conservative finite volume scheme

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

Numerical update

Time-explicit conservative finite volume scheme

$$\mathcal{H}^{(\Delta t)}: \; \mathbf{Q}_{jk}(t+\Delta t) = \mathbf{Q}_{jk}(t) - \frac{\Delta t}{\Delta x_1}\left(\mathbf{F}^1_{j+\frac{1}{2},k} - \mathbf{F}^1_{j-\frac{1}{2},k}\right) - \frac{\Delta t}{\Delta x_2}\left(\mathbf{F}^2_{j,k+\frac{1}{2}} - \mathbf{F}^2_{j,k-\frac{1}{2}}\right)$$

`UpdateLevel(`l`)`

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

$$\mathbf{Q}(G^s_{l,m}, t) \stackrel{\mathcal{H}^{(\Delta t_l)}}{\longrightarrow} \mathbf{Q}(G_{l,m}, t + \Delta t_l) \, , \mathbf{F}^n(\bar{G}_{l,m}, t)$$

Numerical update

Time-explicit conservative finite volume scheme

$$\mathcal{H}^{(\Delta t)}: \; \mathbf{Q}_{jk}(t+\Delta t) = \mathbf{Q}_{jk}(t) - \frac{\Delta t}{\Delta x_1}\left(\mathbf{F}^1_{j+\frac{1}{2},k} - \mathbf{F}^1_{j-\frac{1}{2},k}\right) - \frac{\Delta t}{\Delta x_2}\left(\mathbf{F}^2_{j,k+\frac{1}{2}} - \mathbf{F}^2_{j,k-\frac{1}{2}}\right)$$

`UpdateLevel(`l`)`

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

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

`If level` $l+1$ `exists`

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

Numerical update

Time-explicit conservative finite volume scheme

$$\mathcal{H}^{(\Delta t)}: \; \mathbf{Q}_{jk}(t+\Delta t) = \mathbf{Q}_{jk}(t) - \frac{\Delta t}{\Delta x_1}\left(\mathbf{F}^1_{j+\frac{1}{2},k} - \mathbf{F}^1_{j-\frac{1}{2},k}\right) - \frac{\Delta t}{\Delta x_2}\left(\mathbf{F}^2_{j,k+\frac{1}{2}} - \mathbf{F}^2_{j,k-\frac{1}{2}}\right)$$

`UpdateLevel(`l`)`

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

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

    `If level` $l>0$

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

    `If level` $l+1$ `exists`

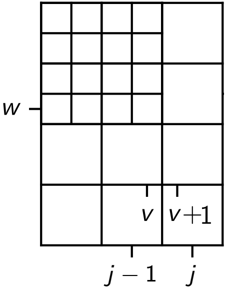
      `Init` $\delta\mathbf{F}^{n,l+1}$ `with` $\mathbf{F}^n(\bar{G}_{l,m}\cap\partial G_{l+1},t)$

Conservative flux correction

Example: Cell j, k

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

Correction pass:



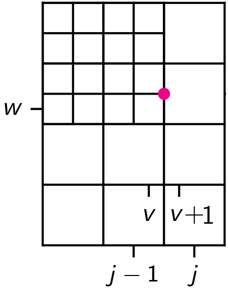
Conservative flux correction

Example: Cell $j,\ k$

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

Correction pass:

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



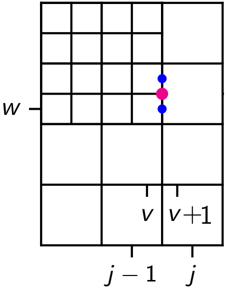
Conservative flux correction

Example: Cell j, k

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

Correction pass:

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



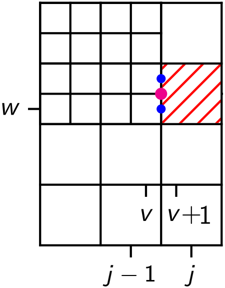
Conservative flux correction

Example: Cell j, k

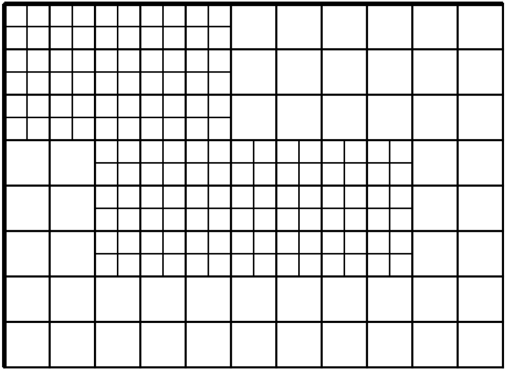
$$\check{\mathbf{Q}}^l_{jk}(t+\Delta t_l) = \mathbf{Q}^l_{jk}(t) - \frac{\Delta t_l}{\Delta x_{1,l}}\left(\mathbf{F}^{1,l}_{j+\frac{1}{2},k} - \frac{1}{r^2_{l+1}}\sum_{\kappa=0}^{r_{l+1}-1}\sum_{\iota=0}^{r_{l+1}-1}\mathbf{F}^{1,l+1}_{v+\frac{1}{2},w+\iota}(t+\kappa\Delta t_{l+1})\right) - \frac{\Delta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

Correction pass:

1. $\delta\mathbf{F}^{1,l+1}_{j-\frac{1}{2},k} := -\mathbf{F}^{1,l}_{j-\frac{1}{2},k}$
2. $\delta\mathbf{F}^{1,l+1}_{j-\frac{1}{2},k} := \delta\mathbf{F}^{1,l+1}_{j-\frac{1}{2},k} + \frac{1}{r^2_{l+1}}\sum_{\iota=0}^{r_{l+1}-1}\mathbf{F}^{1,l+1}_{v+\frac{1}{2},w+\iota}(t+\kappa\Delta t_{l+1})$
3. $\check{\mathbf{Q}}^l_{jk}(t+\Delta t_l) := \mathbf{Q}^l_{jk}(t+\Delta t_l) + \frac{\Delta t_l}{\Delta x_{1,l}}\,\delta\mathbf{F}^{1,l+1}_{j-\frac{1}{2},k}$

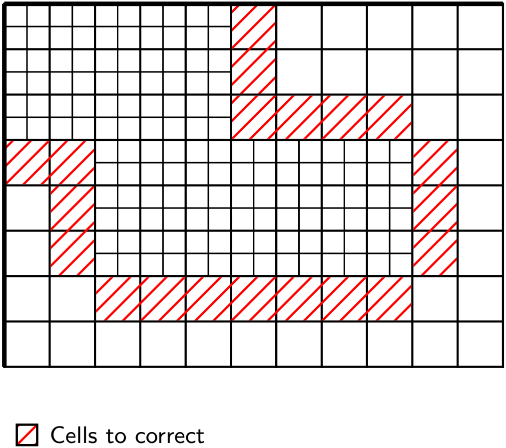


Conservative flux correction II



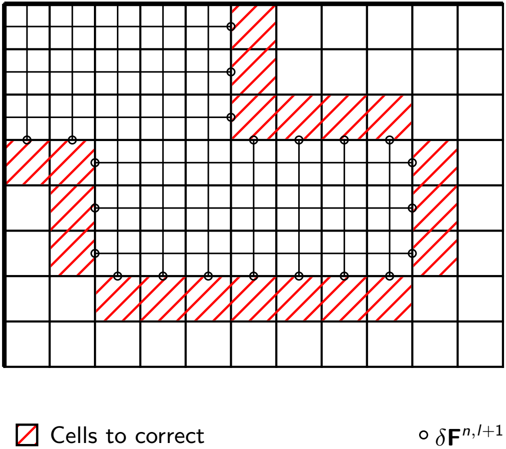
Conservative flux correction II

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



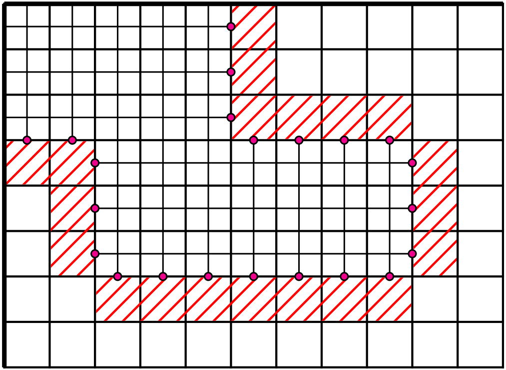
Conservative flux correction II

- Level l cells needing correction $(G_{l+1}^{r_{l+1}} \backslash G_{l+1}) \cap G_l$
- Corrections $\delta \mathbf{F}^{n,l+1}$ stored on level $l+1$ along ∂G_{l+1} (lower-dimensional data coarsened by r_{l+1})



Conservative flux correction II

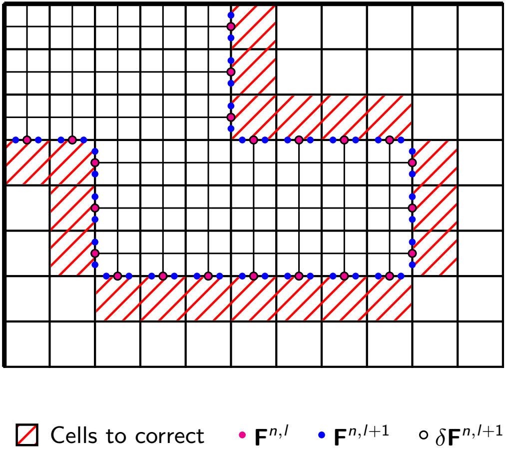
- Level l cells needing correction $(G_{l+1}^{r_{l+1}} \backslash G_{l+1}) \cap G_l$
- Corrections $\delta \mathbf{F}^{n,l+1}$ stored on level $l+1$ along ∂G_{l+1} (lower-dimensional data coarsened by r_{l+1})
- Init $\delta \mathbf{F}^{n,l+1}$ with level l fluxes $\mathbf{F}^{n,l}(\bar{G}_l \cap \partial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



Cells to correct • $\mathbf{F}^{n,l}$ ∘ $\delta \mathbf{F}^{n,l+1}$

Conservative flux correction II

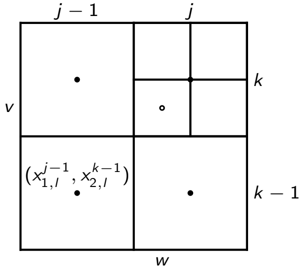
- Level l cells needing correction $(G_{l+1}^{r_{l+1}} \backslash G_{l+1}) \cap G_l$
- Corrections $\delta\mathbf{F}^{n,l+1}$ stored on level $l+1$ along ∂G_{l+1} (lower-dimensional data coarsened by r_{l+1})
- Init $\delta\mathbf{F}^{n,l+1}$ with level l fluxes $\mathbf{F}^{n,l}(\bar{G}_l \cap \partial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



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

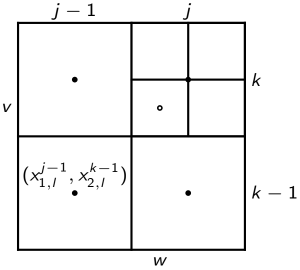
$$\hat{\mathbf{Q}}^l_{jk} := \frac{1}{(r_{l+1})^2} \sum_{\kappa=0}^{r_{l+1}-1} \sum_{\iota=0}^{r_{l+1}-1} \mathbf{Q}^{l+1}_{v+\kappa, w+\iota}$$



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

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



Bilinear interpolation (prolongation):

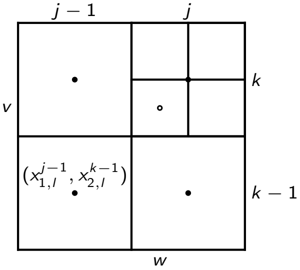
$$\check{\mathbf{Q}}^{l+1}_{vw} := (1-f_1)(1-f_2)\,\mathbf{Q}^l_{j-1,k-1} + f_1(1-f_2)\,\mathbf{Q}^l_{j,k-1} + (1-f_1)f_2\,\mathbf{Q}^l_{j-1,k} + f_1 f_2\,\mathbf{Q}^l_{jk}$$

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

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

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



Bilinear interpolation (prolongation):

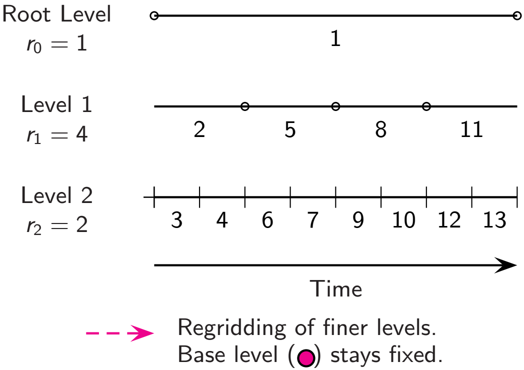
$$\check{\mathbf{Q}}^{l+1}_{vw} := (1-f_1)(1-f_2)\,\mathbf{Q}^l_{j-1,k-1} + f_1(1-f_2)\,\mathbf{Q}^l_{j,k-1} + (1-f_1)f_2\,\mathbf{Q}^l_{j-1,k} + f_1 f_2\,\mathbf{Q}^l_{jk}$$

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

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

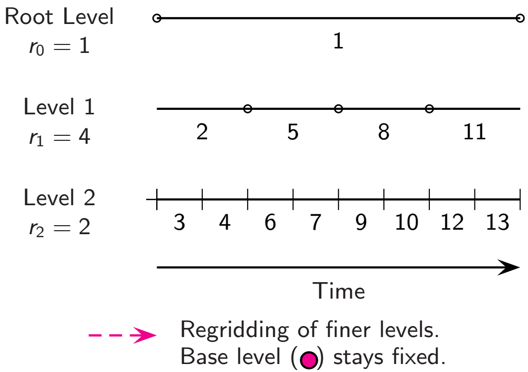
$$\tilde{\mathbf{Q}}^{l+1}(t+\kappa\Delta t_{l+1}) := \left(1 - \frac{\kappa}{r_{l+1}}\right) \check{\mathbf{Q}}^{l+1}(t) + \frac{\kappa}{r_{l+1}} \check{\mathbf{Q}}^{l+1}(t+\Delta t_l) \quad \text{for } \kappa = 0, \ldots r_{l+1}$$

Recursive integration order



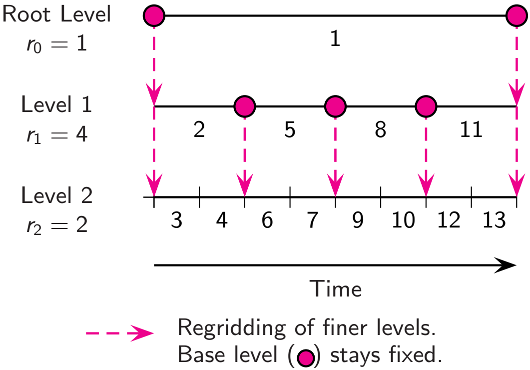
Recursive integration order

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



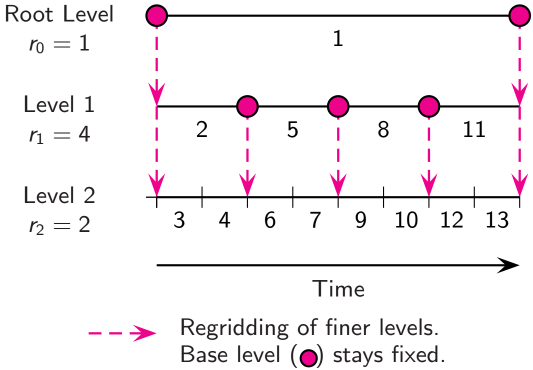
Recursive integration order

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



Recursive integration order

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



The basic recursive algorithm

```
AdvanceLevel(l)

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

      UpdateLevel(l)

      t := t + Δt_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+Δt_l)
          AdvanceLevel(l+1)


      t := t+Δt_l
```

- Recursion

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+Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t+Δt_l) onto Q^l(t+Δt_l)
          Correct Q^l(t+Δt_l) with δF^{l+1}
      t := t+Δt_l
```

- Recursion
- Restriction and flux correction

The basic recursive algorithm

AdvanceLevel(l)

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

- Recursion
- Restriction and flux correction
- Re-organization of hierarchical data

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(t+Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t+Δt_l) onto Q^l(t+Δt_l)
          Correct Q^l(t+Δt_l) with δF^{l+1}
      t := t+Δt_l

Start - Start integration on level 0

     l = 0, r_0 = 1
     AdvanceLevel(l)
```

- Recursion
- Restriction and flux correction
- Re-organization of hierarchical data

The basic recursive algorithm

AdvanceLevel(l)

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

Start - Start integration on level 0

$l=0$, $r_0=1$
AdvanceLevel(l)

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

- Recursion
- Restriction and flux correction
- Re-organization of hierarchical data

Regridding algorithm

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

For $\iota = l_f$ Downto l Do
    Flag N^ι according to $\mathbf{Q}^\iota(t)$

Regridding algorithm

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

`For` $\iota = l_f$ `Downto` l `Do`
`Flag` N^ι `according to` $\mathbf{Q}^\iota(t)$

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

Regridding algorithm

```
Regrid(l) - Regrid all levels ι > l

  For ι = l_f Downto l Do
      Flag N^ι according to Q^ι(t)
      If level ι + 1 exists?
          Flag N^ι below Ğ^(ι+2)
```

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

Regridding algorithm

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

```
For ι = l_f Downto l Do
    Flag N^ι according to Q^ι(t)
    If level ι+1 exists?
        Flag N^ι below Ğ^(ι+2)
    Flag buffer zone on N^ι
```

- Refinement flags: $N^l := \bigcup_m N(\partial G_{l,m})$
- Activate flags below higher levels
- Flag buffer cells of $b > \kappa_r$ cells, κ_r steps between calls of `Regrid(`l`)`

Regridding algorithm

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

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

- Refinement flags: $N^l := \bigcup_m N(\partial G_{l,m})$
- Activate flags below higher levels
- Flag buffer cells of $b > \kappa_r$ cells, κ_r steps between calls of `Regrid(`l`)`
- Special cluster algorithm

Regridding algorithm

```
Regrid(l) - Regrid all levels ι > l

  For ι = l_f Downto l Do
      Flag N^ι according to Q^ι(t)
      If level ι+1 exists?
          Flag N^ι below Ğ^{ι+2}
      Flag buffer zone on N^ι
      Generate Ğ^{ι+1} from N^ι
  Ğ_l := G_l
  For ι = l To l_f Do
      CĞ_ι := G_0 \ Ğ_ι
      Ğ_{ι+1} := Ğ_{ι+1} \ CĞ_ι^1
```

- Refinement flags: $N^l := \bigcup_m N(\partial G_{l,m})$
- Activate flags below higher levels
- Flag buffer cells of $b > \kappa_r$ cells, κ_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 $\iota > l$

For $\iota = l_f$ Downto l Do
  Flag N^ι according to $\mathbf{Q}^\iota(t)$
  If level $\iota+1$ exists?
    Flag N^ι below $\breve{G}^{\iota+2}$
  Flag buffer zone on N^ι
  Generate $\breve{G}^{\iota+1}$ from N^ι
$\breve{G}_l := G_l$
For $\iota = l$ To l_f Do
  $C\breve{G}_\iota := G_0 \backslash \breve{G}_\iota$
  $\breve{G}_{\iota+1} := \breve{G}_{\iota+1} \backslash C\breve{G}_\iota^1$

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, κ_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` $\iota > l$

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

- Creates max. 1 level above l_f, but can remove multiple level if $\check{G}_\iota$ empty (no coarsening!)

Recomposition of data

`Recompose(`l`) - Reorganize all levels` $\iota > l$

`For` $\iota = l + 1$ `To` $l_f + 1$ `Do`
    `Interpolate` $\mathbf{Q}^{\iota-1}(t)$ `onto` $\check{\mathbf{Q}}^{\iota}(t)$

- Creates max. 1 level above l_f, but can remove multiple level if $\check{G}_\iota$ empty (no coarsening!)
- Use spatial interpolation on entire data $\check{\mathbf{Q}}^{\iota}(t)$

Recomposition of data

`Recompose`(l) - `Reorganize all levels` $\iota > l$

`For` $\iota = l + 1$ `To` $l_f + 1$ `Do`
  `Interpolate` $\mathbf{Q}^{\iota-1}(t)$ `onto` $\check{\mathbf{Q}}^{\iota}(t)$
  `Copy` $\mathbf{Q}^{\iota}(t)$ `onto` $\check{\mathbf{Q}}^{\iota}(t)$

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

Recomposition of data

```
Recompose(l) - Reorganize all levels ι > l

  For ι = l + 1 To l_f + 1 Do
      Interpolate Q^{ι−1}(t) onto Q̆^ι(t)
      Copy Q^ι(t) onto Q̆^ι(t)
      Set ghost cells of Q̆^ι(t)
```

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

Recomposition of data

`Recompose(`l`) - Reorganize all levels` $\iota > l$

`For` $\iota = l+1$ `To` $l_f + 1$ `Do`
`Interpolate` $\mathbf{Q}^{\iota-1}(t)$ `onto` $\check{\mathbf{Q}}^{\iota}(t)$
`Copy` $\mathbf{Q}^{\iota}(t)$ `onto` $\check{\mathbf{Q}}^{\iota}(t)$
`Set ghost cells of` $\check{\mathbf{Q}}^{\iota}(t)$
$\mathbf{Q}^{\iota}(t) := \check{\mathbf{Q}}^{\iota}(t)$, $G_\iota := \check{G}_\iota$

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

Clustering by signatures

				x	x	x	x	x	x	6
				x	x	x	x	x	x	6
			x	x	x					3
	x	x	x							3
	x	x								2
	x	x								2
	x	x								2
										0
	x	x								2
	x	x								2
Υ	6	6	2	3	2	2	2	2	2	

Υ Flagged cells per row/column

Δ Second derivative of Υ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_\nu + \Upsilon_{\nu-1}$

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

Clustering by signatures

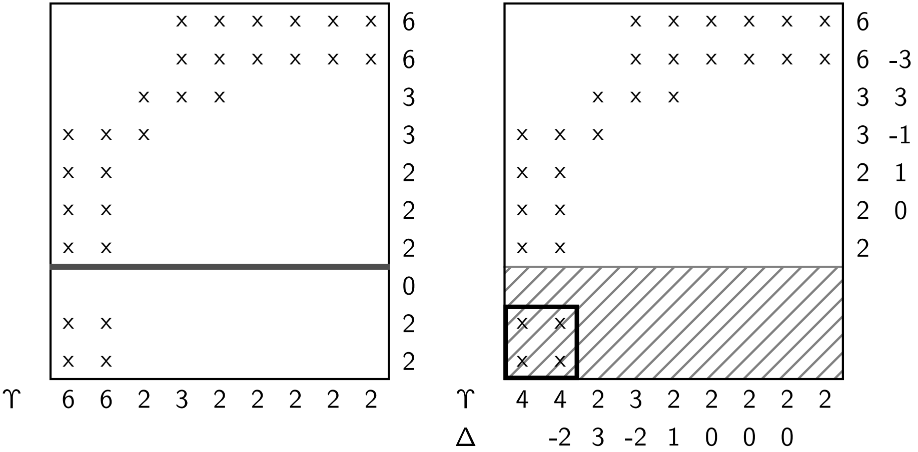
				x	x	x	x	x	x	6
				x	x	x	x	x	x	6
			x	x	x					3
	x	x	x							3
	x	x								2
	x	x								2
	x	x								2
										0
	x	x								2
	x	x								2
Υ	6	6	2	3	2	2	2	2	2	

Υ Flagged cells per row/column

Δ Second derivative of Υ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_\nu + \Upsilon_{\nu-1}$

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

Clustering by signatures

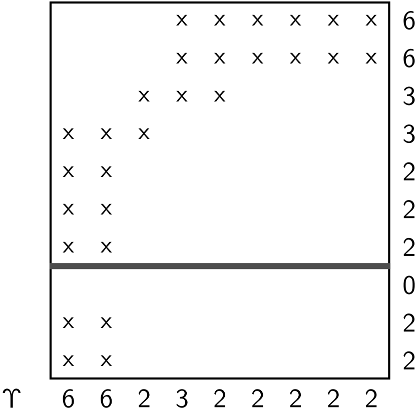


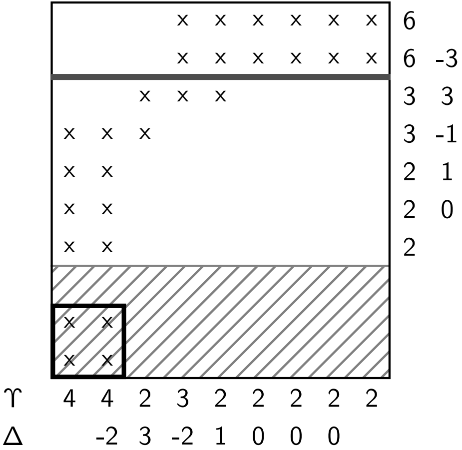
Υ Flagged cells per row/column

Δ Second derivative of Υ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_\nu + \Upsilon_{\nu-1}$

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

Clustering by signatures

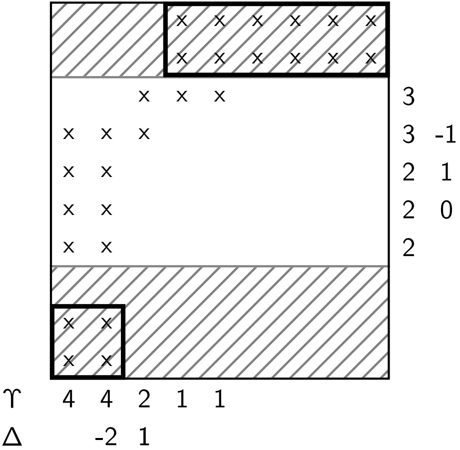




Υ Flagged cells per row/column

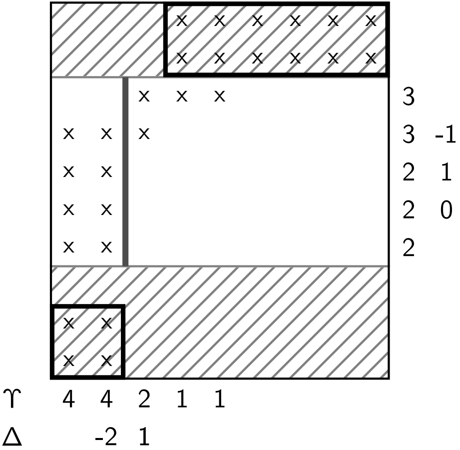
Δ Second derivative of Υ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_{\nu} + \Upsilon_{\nu-1}$

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



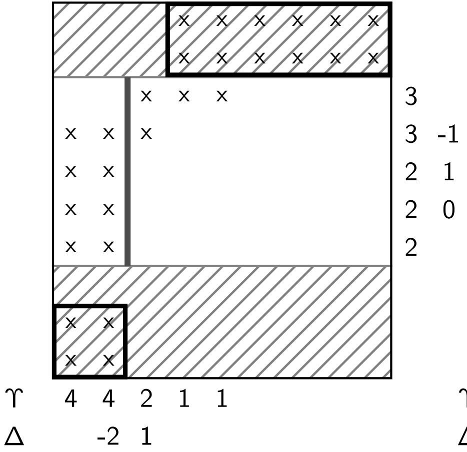
Recursive generation of $\breve{G}_{l,m}$

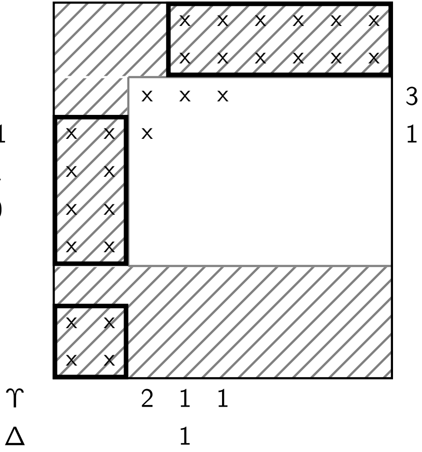
1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$



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

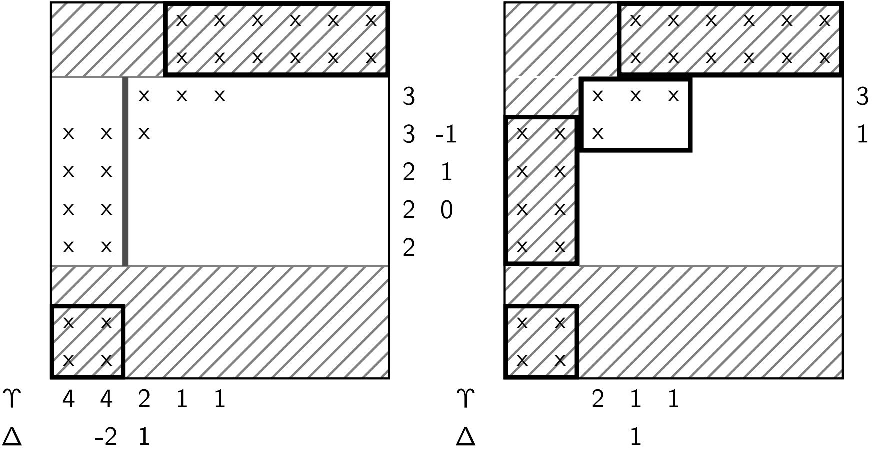
1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$





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

1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$



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

1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$

Refinement criteria

Scaled gradient of scalar quantity w

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

Refinement criteria

Scaled gradient of scalar quantity w

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

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

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

Refinement criteria

Scaled gradient of scalar quantity w

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

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

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

For $\mathbf{q}$ smooth after 2 steps Δt

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

Refinement criteria

Scaled gradient of scalar quantity w

$$|w(\mathbf{Q}_{j+1,k})-w(\mathbf{Q}_{jk})| > \epsilon_w \,, \; |w(\mathbf{Q}_{j,k+1})-w(\mathbf{Q}_{jk})| > \epsilon_w \,, \; |w(\mathbf{Q}_{j+1,k+1})-w(\mathbf{Q}_{jk})| > \epsilon_w$$

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

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

For $\mathbf{q}$ smooth after 2 steps Δt

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

and after 1 step with $2\Delta t$

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

Refinement criteria

Scaled gradient of scalar quantity w

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

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

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

For $\mathbf{q}$ smooth after 2 steps Δt

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

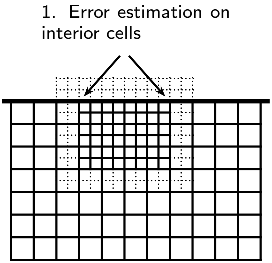
and after 1 step with $2\Delta t$

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

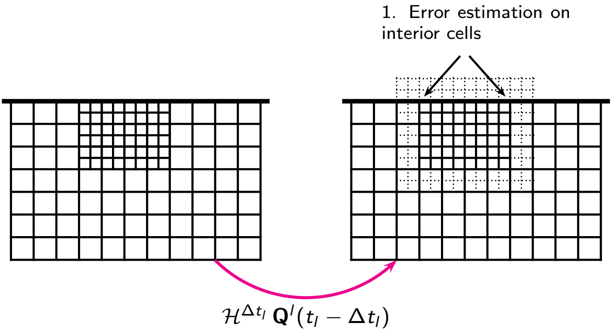
Gives

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

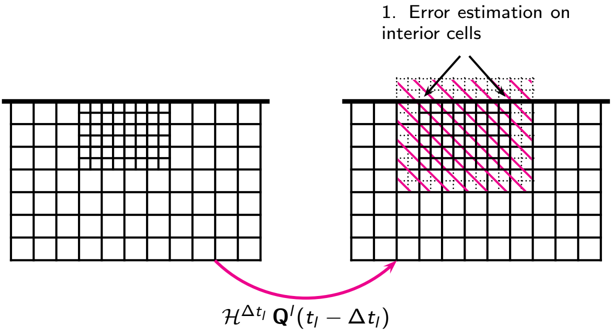
Heuristic error estimation for FV methods



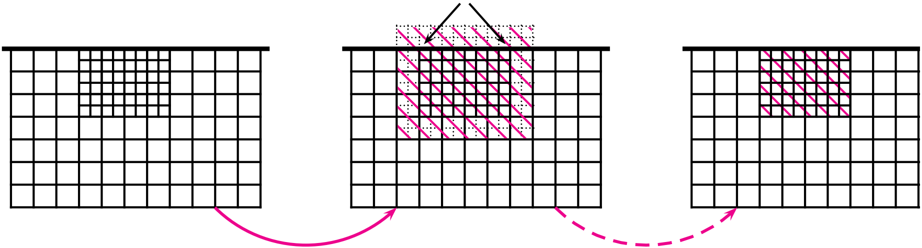
Heuristic error estimation for FV methods



Heuristic error estimation for FV methods



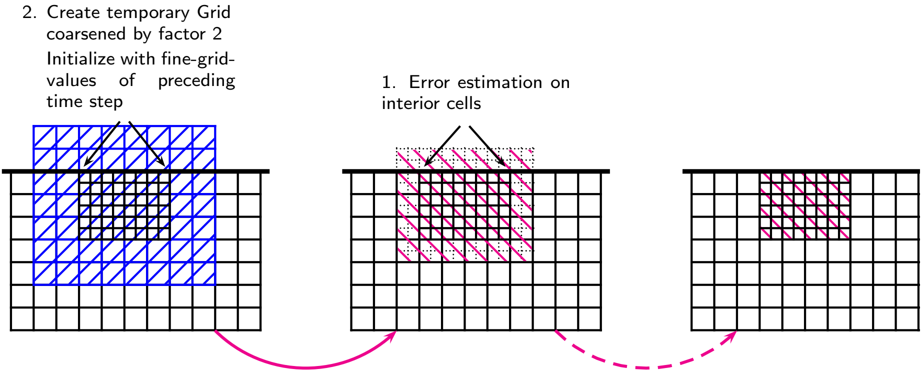
Heuristic error estimation for FV methods



$$\mathcal{H}^{\Delta t_l}\,\mathbf{Q}^l(t_l - \Delta t_l)$$

$$\mathcal{H}^{\Delta t_l}(\mathcal{H}^{\Delta t_l}\,\mathbf{Q}^l(t_l - \Delta t_l)) = \mathcal{H}_2^{\Delta t_l}\,\mathbf{Q}^l(t_l - \Delta t_l)$$

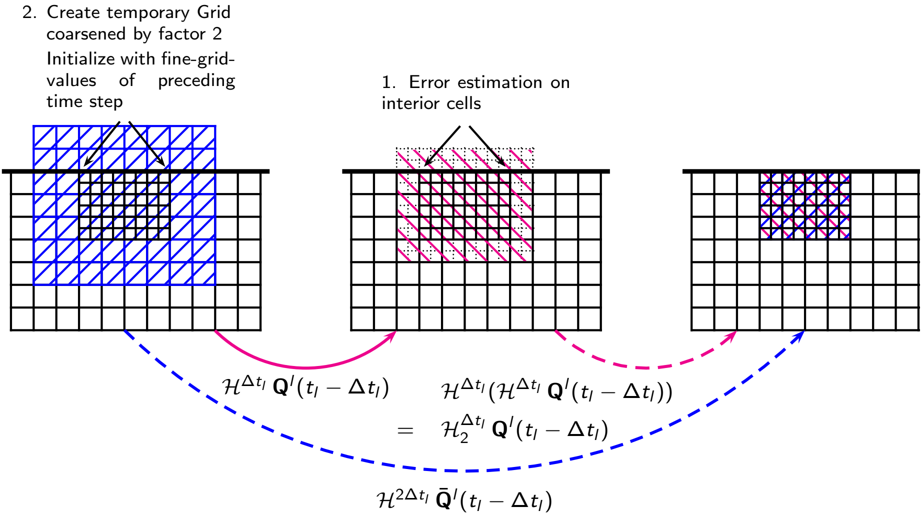
Heuristic error estimation for FV methods



$$\mathcal{H}^{\Delta t_l}\,\mathbf{Q}^l(t_l-\Delta t_l)$$

$$\begin{aligned}&\mathcal{H}^{\Delta t_l}(\mathcal{H}^{\Delta t_l}\,\mathbf{Q}^l(t_l-\Delta t_l))\\ =\;&\mathcal{H}_2^{\Delta t_l}\,\mathbf{Q}^l(t_l-\Delta t_l)\end{aligned}$$

Heuristic error estimation for FV methods

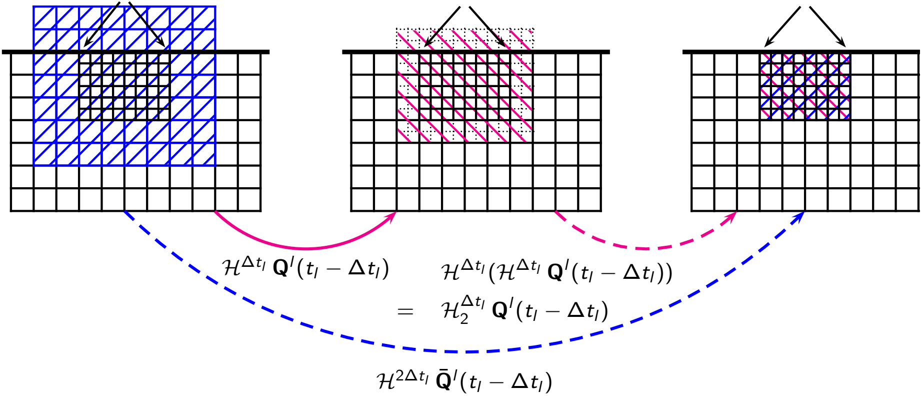


Heuristic error estimation for FV methods

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

1. Error estimation on interior cells

3. Compare temporary solutions



Usage of heuristic error estimation

Current solution integrated tentatively 1 step with Δt_l and coarsened

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

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

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

Usage of heuristic error estimation

Current solution integrated tentatively 1 step with Δt_l and coarsened

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

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

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

Local error estimation of scalar quantity w

$$\tau^w_{jk} := \frac{|w(\bar{\mathcal{Q}}_{jk}(t+\Delta t)) - w(\mathcal{Q}_{jk}(t+\Delta t))|}{2^{o+1}-2}$$

Usage of heuristic error estimation

Current solution integrated tentatively 1 step with Δt_l and coarsened

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

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

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

Local error estimation of scalar quantity w

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

In practice [Deiterding, 2003] use

$$\frac{\tau_{jk}^w}{\max(|w(\mathcal{Q}_{jk}(t + \Delta t))|, S_w)} > \eta_w^r$$

Outline

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

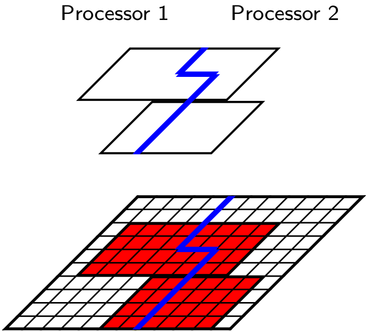
Decomposition of the hierarchical data

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

Parallelization strategies

Decomposition of the hierarchical data

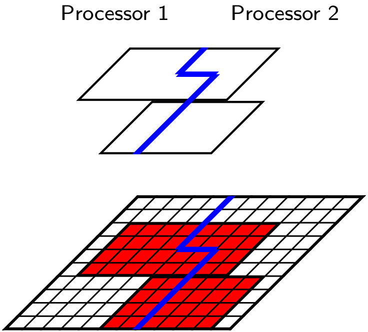
- Distribution of each grid
- Separate distribution of each level, cf. [Rendleman et al., 2000]
- Rigorous domain decomposition
  - Data of all levels resides on same node



Parallelization strategies

Decomposition of the hierarchical data

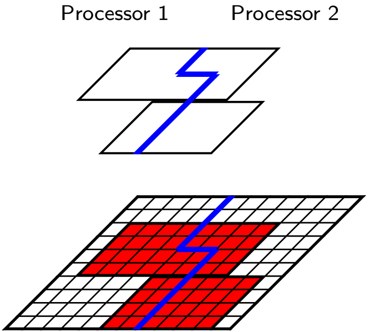
- Distribution of each grid
- Separate distribution of each level, cf. [Rendleman et al., 2000]
- Rigorous domain decomposition
  - Data of all levels resides on same node
  - Grid hierarchy defines unique "floor-plan"



Parallelization strategies

Decomposition of the hierarchical data

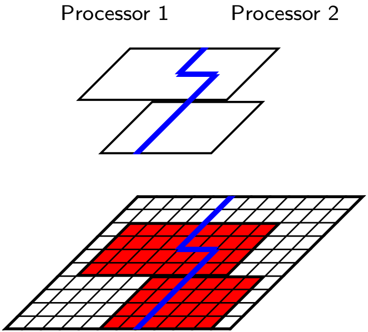
- Distribution of each grid
- Separate distribution of each level, cf. [Rendleman et al., 2000]
- Rigorous domain decomposition
  - Data of all levels resides on same node
  - Grid hierarchy defines unique "floor-plan"
  - Redistribution of data blocks during reorganization of hierarchical data



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

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

Rigorous domain decomposition formalized

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

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

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

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

Higher level domains G_l follow decomposition of root level

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

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

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

Rigorous domain decomposition formalized

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

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

Higher level domains G_l follow decomposition of root level

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

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

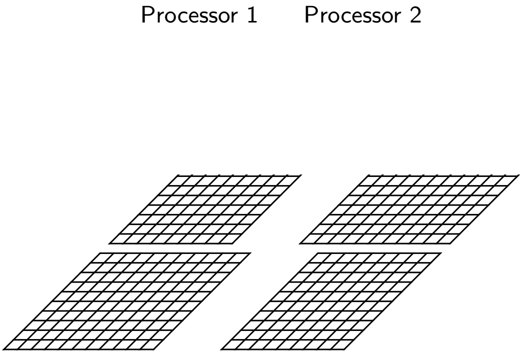
$$\mathcal{W}(\Omega) = \sum_{l=0}^{l_{\max}} \left[\mathcal{N}_l(G_l \cap \Omega) \prod_{\kappa=0}^{l} r_\kappa \right]$$

Equal work distribution necessitates

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

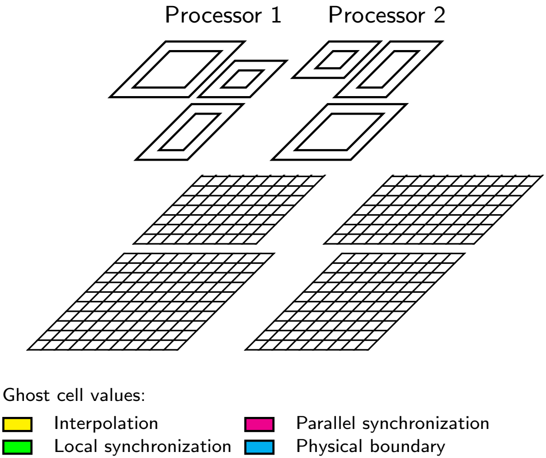
[Deiterding, 2005]

Ghost cell setting



Ghost cell values:

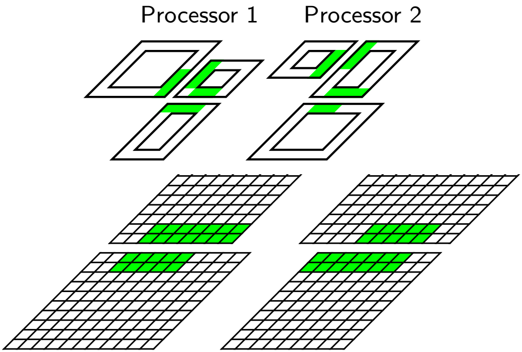
Ghost cell setting



Ghost cell setting

Local synchronization

$$\tilde{S}^{s,p}_{l,m} = \tilde{G}^{s,p}_{l,m} \cap G^p_l$$



Ghost cell values:

- Interpolation
- Local synchronization
- Parallel synchronization
- Physical boundary

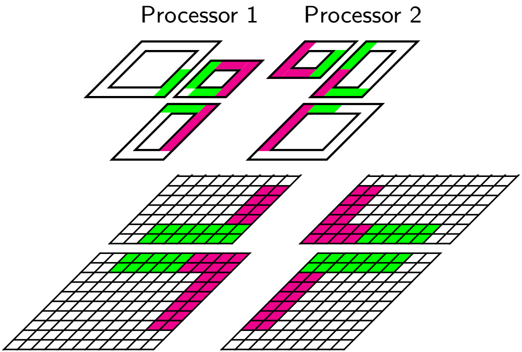
Ghost cell setting

Local synchronization

$$\tilde{S}^{s,p}_{l,m} = \tilde{G}^{s,p}_{l,m} \cap G^p_l$$

Parallel synchronization

$$\tilde{S}^{s,q}_{l,m} = \tilde{G}^{s,p}_{l,m} \cap G^q_l, q \neq p$$



Ghost cell values:

- Interpolation
- Local synchronization
- Parallel synchronization
- Physical boundary

Ghost cell setting

Local synchronization

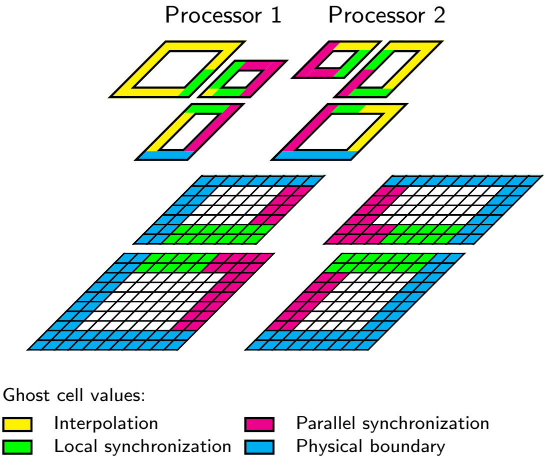
$$\tilde{S}^{s,p}_{l,m} = \tilde{G}^{s,p}_{l,m} \cap G^p_l$$

Parallel synchronization

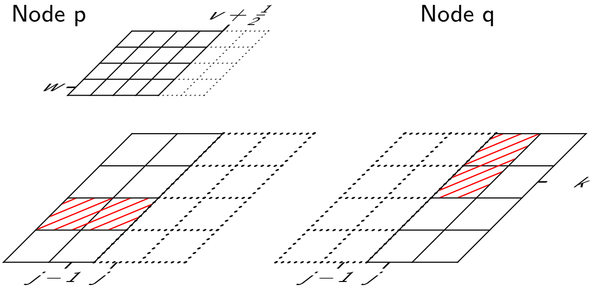
$$\tilde{S}^{s,q}_{l,m} = \tilde{G}^{s,p}_{l,m} \cap G^q_l,\ q \neq p$$

Interpolation and physical boundary conditions remain strictly local

- Scheme $\mathcal{H}^{(\Delta t_l)}$ evaluated locally
- Restriction and propolongation local

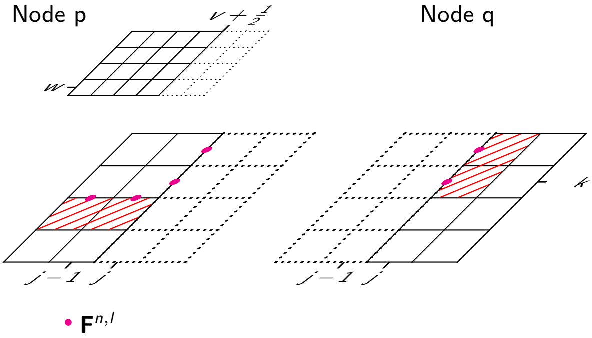


Parallel flux correction



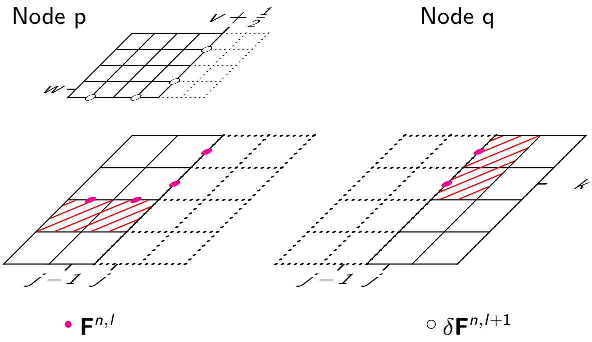
Parallel flux correction

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



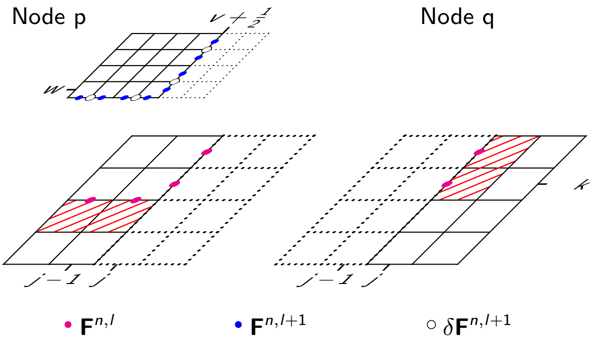
Parallel flux correction

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



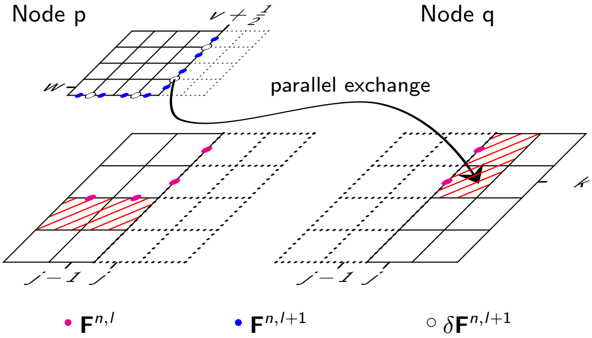
Parallel flux correction

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



Parallel flux correction

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



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+Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t+Δt_l) onto Q^l(t+Δt_l)
          Correct Q^l(t+Δt_l) with δF^{l+1}
      t := t+Δt_l
```

```
UpdateLevel(l)

  For all m = 1 To M_l Do
      Q(G^s_{l,m}, t) --H^(Δt_l)--> Q(G_{l,m}, t+Δt_l), F^n(Ḡ_{l,m}, t)
      If level l > 0
          Add F^n(∂G_{l,m}, t) to δF^{n,l}
      If level l+1 exists
          Init δF^{n,l+1} with F^n(Ḡ_{l,m} ∩ ∂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+Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t+Δt_l) onto Q^l(t+Δt_l)
          Correct Q^l(t+Δt_l) with δF^{l+1}
      t := t+Δt_l

UpdateLevel(l)

  For all m = 1 To M_l Do
      Q(G^s_{l,m}, t) --H^(Δt_l)--> Q(G_{l,m}, t+Δt_l), F^n(Ḡ_{l,m}, t)
      If level l > 0
          Add F^n(∂G_{l,m}, t) to δF^{n,l}
      If level l+1 exists
          Init δF^{n,l+1} with F^n(Ḡ_{l,m} ∩ ∂G_{l+1}, t)
```

- Numerical update strictly local

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+Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t+Δt_l) onto Q^l(t+Δt_l)
          Correct Q^l(t+Δt_l) with δF^{l+1}
      t := t+Δt_l

UpdateLevel(l)

  For all m = 1 To M_l Do
      Q(G^s_{l,m}, t) --H^(Δt_l)--> Q(G_{l,m}, t+Δt_l) , F^n(Ḡ_{l,m}, t)
      If level l > 0
          Add F^n(∂G_{l,m}, t) to δF^{n,l}
      If level l+1 exists
          Init δF^{n,l+1} with F^n(Ḡ_{l,m} ∩ ∂G_{l+1}, t)
```

- Numerical update strictly local
- Inter-level transfer local

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 + Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t + Δt_l) onto Q^l(t + Δt_l)
          Correct Q^l(t + Δt_l) with δF^{l+1}
      t := t + Δt_l

UpdateLevel(l)

  For all m = 1 To M_l Do
      Q(G^s_{l,m}, t) --H^(Δt_l)--> Q(G_{l,m}, t + Δt_l) , F^n(Ḡ_{l,m}, t)
      If level l > 0
          Add F^n(∂G_{l,m}, t) to δF^{n,l}
      If level l+1 exists
          Init δF^{n,l+1} with F^n(Ḡ_{l,m} ∩ ∂G_{l+1}, t)
```

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

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+Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t+Δt_l) onto Q^l(t+Δt_l)
          Correct Q^l(t+Δt_l) with δF^{l+1}
      t := t+Δt_l

UpdateLevel(l)

  For all m = 1 To M_l Do
      Q(G^s_{l,m}, t) --H^(Δt_l)--> Q(G_{l,m}, t+Δt_l), F^n(Ḡ_{l,m}, t)
      If level l > 0
          Add F^n(∂G_{l,m}, t) to δF^{n,l}
      If level l+1 exists
          Init δF^{n,l+1} with F^n(Ḡ_{l,m} ∩ ∂G_{l+1}, t)
```

- Numerical update strictly local
- Inter-level transfer local
- Parallel synchronization
- Application of $\delta\mathbf{F}^{l+1}$ on ∂G_l^q

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+Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t+Δt_l) onto Q^l(t+Δt_l)
          Correct Q^l(t+Δt_l) with δF^{l+1}
      t := t+Δt_l

UpdateLevel(l)

  For all m = 1 To M_l Do
      Q(G^s_{l,m}, t) --H^(Δt_l)--> Q(G_{l,m}, t+Δt_l), F^n(Ḡ_{l,m}, t)
      If level l > 0
          Add F^n(∂G_{l,m}, t) to δF^{n,l}
      If level l+1 exists
          Init δF^{n,l+1} with F^n(Ḡ_{l,m} ∩ ∂G_{l+1}, t)
```

- Numerical update strictly local
- Inter-level transfer local
- Parallel synchronization
- Application of $\delta\mathbf{F}^{l+1}$ on ∂G_l^q

Regridding algorithm in parallel

```
Regrid(l) - Regrid all levels ι > l

  For ι = l_f Downto l Do
      Flag N^ι according to Q^ι(t)
      If level ι+1 exists?
          Flag N^ι below Ğ^{ι+2}
      Flag buffer zone on N^ι
      Generate Ğ^{ι+1} from N^ι
  Ğ_l := G_l
  For ι = l To l_f Do
      CĞ_ι := G_0 \ Ğ_ι
      Ğ_{ι+1} := Ğ_{ι+1} \ CĞ_ι^1

  Recompose(l)
```

Regridding algorithm in parallel

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

For $\iota = l_f$ Downto l Do
  Flag N^ι according to $\mathbf{Q}^\iota(t)$
  If level $\iota + 1$ exists?
    Flag N^ι below $\breve{G}^{\iota+2}$
  Flag buffer zone on N^ι
  Generate $\breve{G}^{\iota+1}$ from N^ι
$\breve{G}_l := G_l$
For $\iota = l$ To l_f Do
  $C\breve{G}_\iota := G_0 \backslash \breve{G}_\iota$
  $\breve{G}_{\iota+1} := \breve{G}_{\iota+1} \backslash C\breve{G}_\iota^1$

Recompose(l)

Regridding algorithm in parallel

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

    For $\iota = l_f$ Downto l Do
        Flag N^ι according to $\mathbf{Q}^\iota(t)$
        If level $\iota+1$ exists?
            Flag N^ι below $\breve{G}^{\iota+2}$
        Flag buffer zone on N^ι
        Generate $\breve{G}^{\iota+1}$ from N^ι
    $\breve{G}_l := G_l$
    For $\iota = l$ To l_f Do
        $C\breve{G}_\iota := G_0 \backslash \breve{G}_\iota$
        $\breve{G}_{\iota+1} := \breve{G}_{\iota+1} \backslash C\breve{G}_\iota^1$

    Recompose(l)

- 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 $\iota = l_f$ Downto l Do
    Flag N^ι according to $\mathbf{Q}^\iota(t)$
    If level $\iota + 1$ exists?
        Flag N^ι below $\breve{G}^{\iota+2}$
    Flag buffer zone on N^ι
    Generate $\breve{G}^{\iota+1}$ from N^ι
$\breve{G}_l := G_l$
For $\iota = l$ To l_f Do
    $C\breve{G}_\iota := G_0 \backslash \breve{G}_\iota$
    $\breve{G}_{\iota+1} := \breve{G}_{\iota+1} \backslash C\breve{G}_\iota^1$

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 $\breve{G}^{\iota+1}$

Regridding algorithm in parallel

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

```
For ι = l_f Downto l Do
    Flag N^ι according to Q^ι(t)
    If level ι+1 exists?
        Flag N^ι below Ğ^{ι+2}
    Flag buffer zone on N^ι
    Generate Ğ^{ι+1} from N^ι
Ğ_l := G_l
For ι = l To l_f Do
    CĞ_ι := G_0 \ Ğ_ι
    Ğ_{ι+1} := Ğ_{ι+1} \ CĞ_ι^1
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 $\breve{G}^{\iota+1}$

Regridding algorithm in parallel

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

For $\iota = l_f$ Downto l Do
    Flag N^ι according to $\mathbf{Q}^\iota(t)$
    If level $\iota + 1$ exists?
        Flag N^ι below $\breve{G}^{\iota+2}$
    Flag buffer zone on N^ι
    Generate $\breve{G}^{\iota+1}$ from N^ι
$\breve{G}_l := G_l$
For $\iota = l$ To l_f Do
    $C\breve{G}_\iota := G_0 \backslash \breve{G}_\iota$
    $\breve{G}_{\iota+1} := \breve{G}_{\iota+1} \backslash C\breve{G}_\iota^1$

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 $\breve{G}^{\iota+1}$

Recomposition algorithm in parallel

Recompose(l) - Reorganize all levels

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

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

Copy $\mathbf{Q}^{\iota}(t)$ onto $\check{\mathbf{Q}}^{\iota}(t)$
Set ghost cells of $\check{\mathbf{Q}}^{\iota}(t)$
$\mathbf{Q}^{\iota}(t) := \check{\mathbf{Q}}^{\iota}(t)$
$G_\iota := \check{G}_\iota$

Recomposition algorithm in parallel

```
Recompose(l) - Reorganize all levels

    Generate G_0^p from {G_0, ..., G_l, Ğ_{l+1}, ..., Ğ_{l_f+1}}
    For ι = 0 To l_f + 1 Do

            Interpolate Q^{ι-1}(t) onto Q̌^ι(t)

            Copy Q^ι(t) onto Q̌^ι(t)
            Set ghost cells of Q̌^ι(t)
            Q^ι(t) := Q̌^ι(t)
        G_ι^p := Ğ_ι^p ,  G_ι := ⋃_p G_ι^p
```

Recompose(l) - Reorganize all levels

Generate G_0^p from $\{G_0, \ldots, G_l, \breve{G}_{l+1}, \ldots, \breve{G}_{l_f+1}\}$
For $\iota = 0$ To $l_f + 1$ Do

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

Copy $\mathbf{Q}^{\iota}(t)$ onto $\breve{\mathbf{Q}}^{\iota}(t)$
Set ghost cells of $\breve{\mathbf{Q}}^{\iota}(t)$
$\mathbf{Q}^{\iota}(t) := \breve{\mathbf{Q}}^{\iota}(t)$
$G_\iota^p := \breve{G}_\iota^p$, $G_\iota := \bigcup_p G_\iota^p$

- 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, Ğ_{l+1}, ..., Ğ_{l_f+1}}
   For ι = 0 To l_f + 1 Do
        If ι > l
             Ğ_ι^p := Ğ_ι ∩ G_0^p
             Interpolate Q^{ι-1}(t) onto Q̌^ι(t)

             Copy Q^ι(t) onto Q̌^ι(t)
             Set ghost cells of Q̌^ι(t)
             Q^ι(t) := Q̌^ι(t)
        G_ι^p := Ğ_ι^p,  G_ι := ⋃_p G_ι^p
```

- 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 $\iota \le l$, redistribute additionally

Recomposition algorithm in parallel

```
Recompose(l) - Reorganize all levels

  Generate G_0^p from {G_0, ..., G_l, Ğ_{l+1}, ..., Ğ_{l_f+1}}
  For ι = 0 To l_f + 1 Do
      If ι > l
          Ğ_ι^p := Ğ_ι ∩ G_0^p
          Interpolate Q^{ι-1}(t) onto Q̆^ι(t)
      else
          Ğ_ι^p := G_ι ∩ G_0^p
          If ι > 0
              Copy δF^{n,ι} onto δF̆^{n,ι}
              δF^{n,ι} := δF̆^{n,ι}

          Copy Q^ι(t) onto Q̆^ι(t)
          Set ghost cells of Q̆^ι(t)
          Q^ι(t) := Q̆^ι(t)
      G_ι^p := Ğ_ι^p,  G_ι := ⋃_p G_ι^p
```

- 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 $\iota \le l$, redistribute additionally
  - Flux corrections $\delta \mathbf{F}^{n,\iota}$

Recomposition algorithm in parallel

Recompose(l) - Reorganize all levels

Generate G_0^p from $\{G_0, \ldots, G_l, \breve{G}_{l+1}, \ldots, \breve{G}_{l_f+1}\}$
For $\iota = 0$ To $l_f + 1$ Do
  If $\iota > l$
    $\breve{G}_\iota^p := \breve{G}_\iota \cap G_0^p$
    Interpolate $\mathbf{Q}^{\iota-1}(t)$ onto $\breve{\mathbf{Q}}^\iota(t)$
  else
    $\breve{G}_\iota^p := G_\iota \cap G_0^p$
    If $\iota > 0$
      Copy $\delta\mathbf{F}^{n,\iota}$ onto $\delta\breve{\mathbf{F}}^{n,\iota}$
      $\delta\mathbf{F}^{n,\iota} := \delta\breve{\mathbf{F}}^{n,\iota}$
  If $\iota \geq l$ then $\kappa_\iota = 0$ else $\kappa_\iota = 1$
  For $\kappa = 0$ To κ_ι Do
    Copy $\mathbf{Q}^\iota(t + \kappa\Delta t_\iota)$ onto $\breve{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
    Set ghost cells of $\breve{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
    $\mathbf{Q}^\iota(t + \kappa\Delta t_\iota) := \breve{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
  $G_\iota^p := \breve{G}_\iota^p$, $G_\iota := \bigcup_p G_\iota^p$

- 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 $\delta\mathbf{F}^{n,\iota}$
  - Already updated time level $\mathbf{Q}^\iota(t + \kappa\Delta t_\iota)$

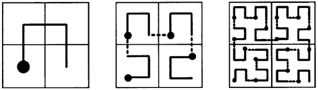
Recomposition algorithm in parallel

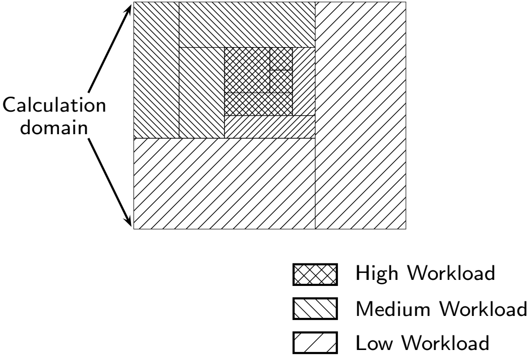
```
Recompose(l) - Reorganize all levels
```

Generate G_0^p from $\{G_0, ..., G_l, \breve{G}_{l+1}, ..., \breve{G}_{l_f+1}\}$
For $\iota = 0$ To $l_f + 1$ Do
  If $\iota > l$
    $\breve{G}_\iota^p := \breve{G}_\iota \cap G_0^p$
    Interpolate $\mathbf{Q}^{\iota-1}(t)$ onto $\breve{\mathbf{Q}}^\iota(t)$
  else
    $\breve{G}_\iota^p := G_\iota \cap G_0^p$
    If $\iota > 0$
      Copy $\delta\mathbf{F}^{n,\iota}$ onto $\delta\breve{\mathbf{F}}^{n,\iota}$
      $\delta\mathbf{F}^{n,\iota} := \delta\breve{\mathbf{F}}^{n,\iota}$
  If $\iota \geq l$ then $\kappa_\iota = 0$ else $\kappa_\iota = 1$
  For $\kappa = 0$ To κ_ι Do
    Copy $\mathbf{Q}^\iota(t + \kappa\Delta t_\iota)$ onto $\breve{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
    Set ghost cells of $\breve{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
    $\mathbf{Q}^\iota(t + \kappa\Delta t_\iota) := \breve{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
  $G_\iota^p := \breve{G}_\iota^p$, $G_\iota := \bigcup_p G_\iota^p$

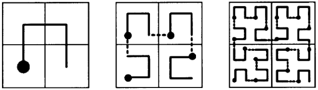
- Global redistribution can also be required when regridding higher levels and $G_0, ..., G_l$ do not change (drawback of domain decomposition)
- When $\iota > l$ do nothing special
- For $\iota \leq l$, redistribute additionally
  - Flux corrections $\delta\mathbf{F}^{n,\iota}$
  - Already updated time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

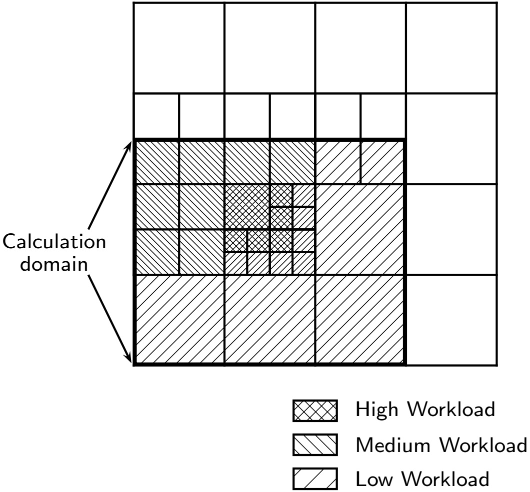
Space-filling curve algorithm



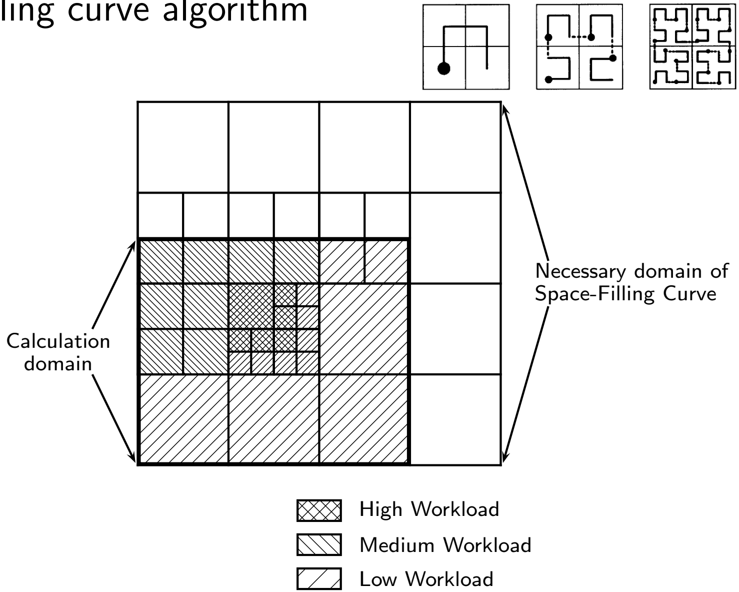


Space-filling curve algorithm

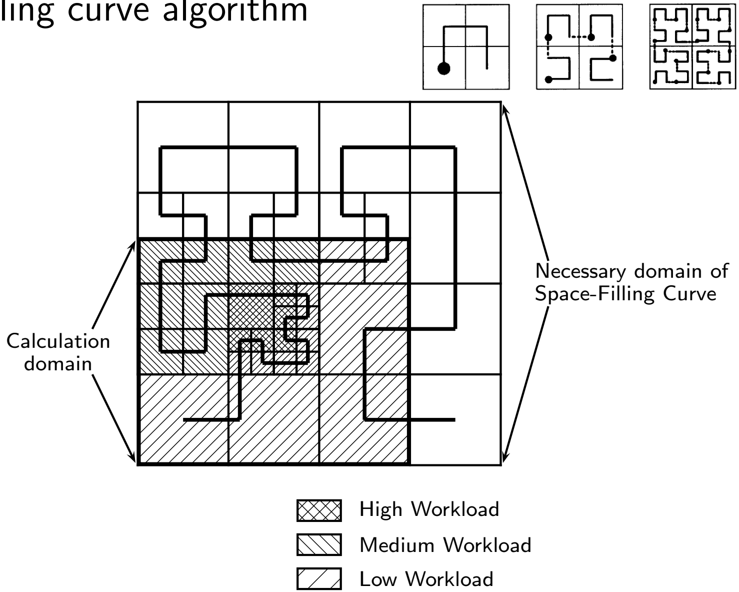




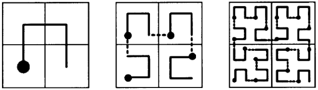
Space-filling curve algorithm

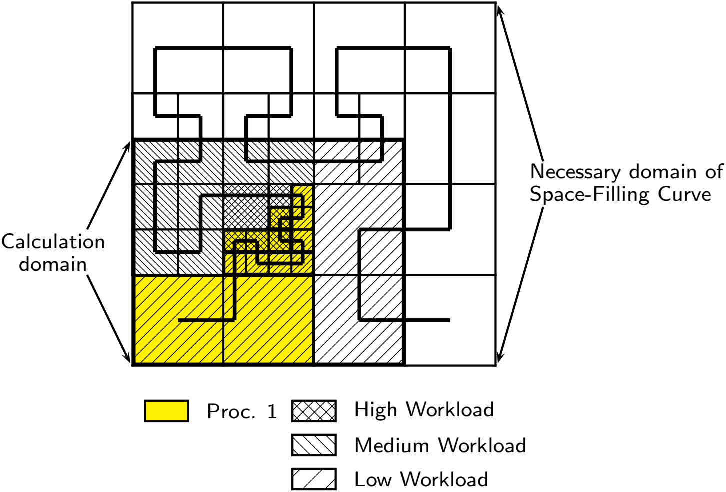


Space-filling curve algorithm

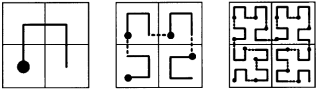


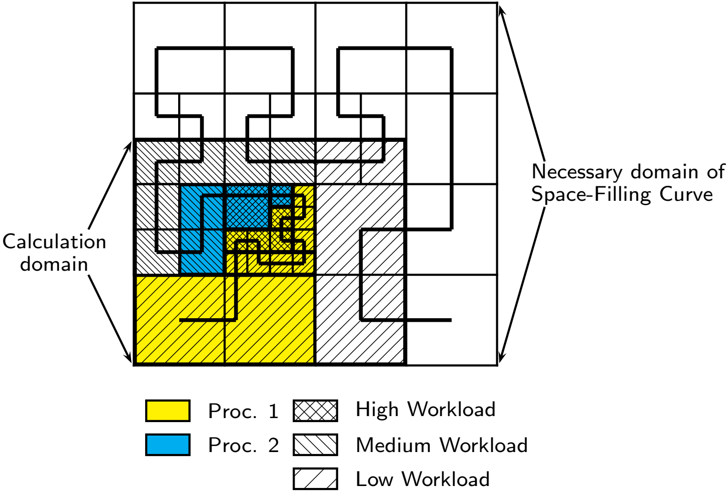
Space-filling curve algorithm



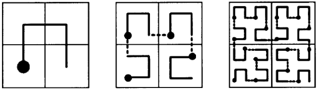


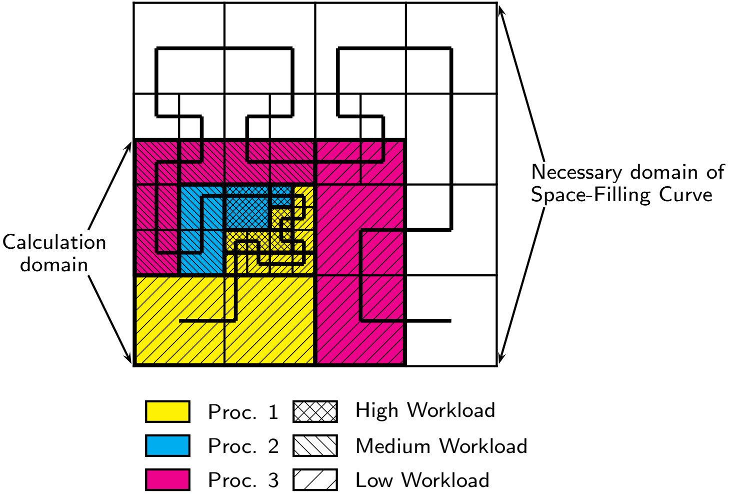
Space-filling curve algorithm





Space-filling curve algorithm



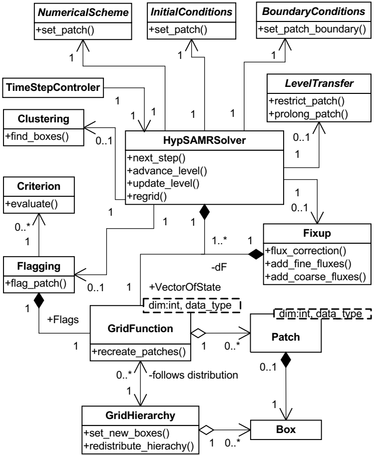


Overview

- "Adaptive Mesh Refinement in Object-oriented C++"
- $\sim 46,000$ LOC for C++ SAMR kernel, $\sim 140,000$ 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
- $\sim 430,000$ 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]

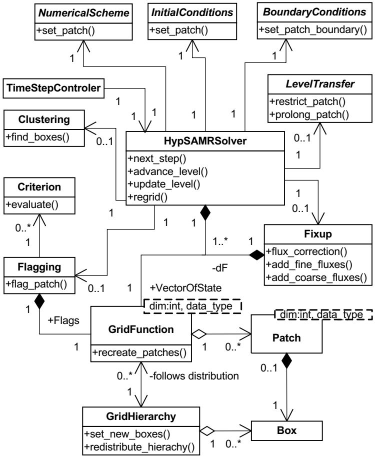
UML design of AMROC

- Classical framework approach with generic main program in C++



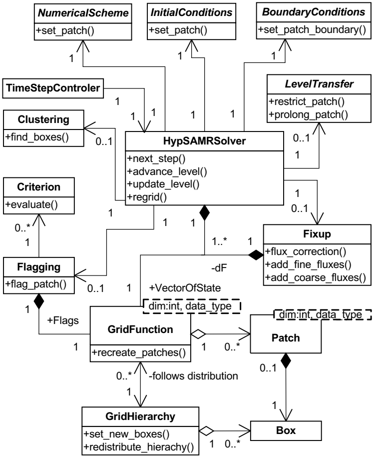
UML design of AMROC

- 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



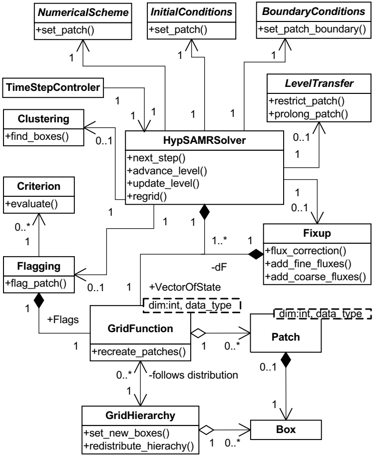
UML design of AMROC

- 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



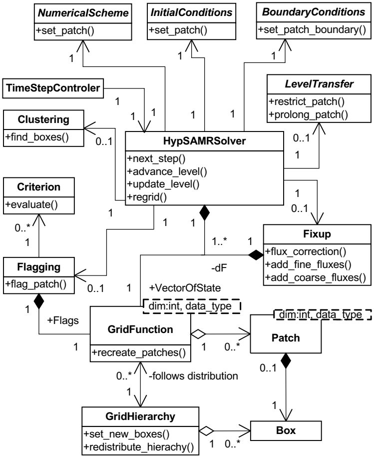
UML design of AMROC

- 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 $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin the hierarchy

Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin 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 $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin 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 $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin 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 topogical relations between lists of Patch objects to implement sychronization, restriction, prolongation, re-distribution

Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin 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 topogical relations between lists of Patch objects to implement sychronization, 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<Type, dim>**: 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<Scalar, length>**: Vector of state is usually Vector<double, N>

  `code/amroc/doc/html/hds/classVector.html`

Hierarchical data structures - II

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

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

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

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

- **GridHierarchy<Type, dim>**: Uses sets of GridBoxList to organize topology of the hierarchy. All GridFunction<Type, dim>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

[Bell et al., 1994] Bell, J., Berger, M., Saltzman, J., and Welcome, M. (1994). Three-dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM J. Sci. Comp.*, 15(1):127–138.

[Berger, 1982] Berger, M. (1982). *Adaptive mesh refinement for hyperbolic differential equations*. PhD thesis, Stanford University. Report No. STAN-CS-82-924.

[Berger, 1986] Berger, M. (1986). Data structures for adaptive grid generation. *SIAM J. Sci. Stat. Comput.*, 7(3):904–916.

[Berger and Colella, 1988] Berger, M. and Colella, P. (1988). Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82:64–84.

[Berger and LeVeque, 1998] Berger, M. and LeVeque, R. (1998). Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems. *SIAM J. Numer. Anal.*, 35(6):2298–2316.

[Berger and Oliger, 1984] Berger, M. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512.

References II

[Berger and Rigoutsos, 1991] Berger, M. and Rigoutsos, I. (1991). An algorithm for point clustering and grid generation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1278–1286.

[Brown et al., 1997] Brown, D. L., Henshaw, W. D., and Quinlan, D. J. (1997). Overture: An object oriented framework for solving partial differential equations. In *Proc. ISCOPE 1997, appeared in Scientific Computing in Object-Oriented Parallel Environments*, number 1343 in Springer Lecture Notes in Computer Science.

[Deiterding, 2003] Deiterding, R. (2003). *Parallel adaptive simulation of multi-dimensional detonation structures*. PhD thesis, Brandenburgische Technische Universität Cottbus.

[Deiterding, 2005] Deiterding, R. (2005). Construction and application of an AMR algorithm for distributed memory computers. In Plewa, T., Linde, T., and Weirs, V. G., editors, *Adaptive Mesh Refinement - Theory and Applications*, volume 41 of *Lecture Notes in Computational Science and Engineering*, pages 361–372. Springer.

[Deiterding et al., 2007] Deiterding, R., Cirak, F., Mauch, S. P., and Meiron, D. I. (2007). A virtual test facility for simulating detonation- and shock-induced deformation and fracture of thin flexible shells. *Int. J. Multiscale Computational Engineering*, 5(1):47–63.

References III

[Deiterding et al., 2006] Deiterding, R., Radovitzky, R., Mauch, S. P., Noels, L., Cummings, J. C., and Meiron, D. I. (2006). A virtual test facility for the efficient simulation of solid materials under high energy shock-wave loading. *Engineering with Computers*, 22(3-4):325–347.

[Gittings et al., 2008] Gittings, M., Weaver, R., Clover, M., Betlach, T., Byrne, N., Coker, R., Dendy, E., Hueckstaedt, R., New, K., Oakes, R., Rantal, D., and Stefan, R. (2008). The RAGE radiation-hydrodynamics code. *Comput. Sci. Disc.*, 1. doi:10.1088/1749-4699/1/1/015005.

[Hornung et al., 2006] Hornung, R. D., Wissink, A. M., and Kohn, S. H. (2006). Managing complex data and geometry in parallel structured AMR applications. *Engineering with Computers*, 22:181–195.

[MacNeice et al., 2000] MacNeice, P., Olson, K. M., Mobarry, C., deFainchtein, R., and Packer, C. (2000). PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126:330–354.

References IV

[Parashar and Browne, 1997] Parashar, M. and Browne, J. C. (1997). System engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement. In *Structured Adaptive Mesh Refinement Grid Methods*, IMA Volumes in Mathematics and its Applications. Springer.

[Rendleman et al., 2000] Rendleman, C. A., Beckner, V. E., Lijewski, M., Crutchfield, W., and Bell, J. B. (2000). Parallelization of structured, hierarchical adaptive mesh refinement algorithms. *Computing and Visualization in Science*, 3:147–157.