Lecture 6

The AMROC software system

Course *Block-structured Adaptive Finite Volume Methods for Shock-Induced Combustion Simulation*

Ralf Deiterding
German Aerospace Center (DLR)
Institute for Aerodynamics and Flow Technology
Bunsenstr. 10, Göttingen, Germany

E-mail: ralf.deiterding@dlr.de

Outline

Available SAMR software

- Simplified block-based AMR
- General patch-based SAMR

Outline

Outline

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

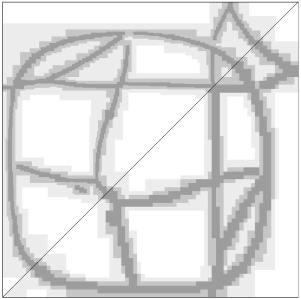
- Overture (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - http://www.overtureframework.org
- AMRClaw within Clawpack [Berger and LeVeque, 1998]
 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

AMROC

- "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

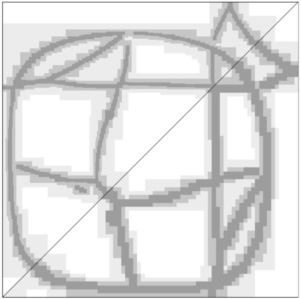
AMROC

- "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
- Right: point explosion in box, 4 level, Euler computation, 7 compute nodes



AMROC

- "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
- Right: point explosion in box, 4 level, Euler computation, 7 compute nodes
- V1.0: http://amroc.sourceforge.net



	l_{max}	Level 0	Level 1	Level 2	Level 3	Level 4
AMROC's DAGH grids/cells	1	43/22500	145/38696			
	2	42/22500	110/48708	283/83688		
	3	36/22500	78/54796	245/109476	582/165784	
	4	41/22500	88/56404	233/123756	476/220540	1017/294828
Original DAGH grids/cells	1	238/22500	125/41312			
	2	494/22500	435/48832	190/105216		
	3	695/22500	650/55088	462/133696	185/297984	
	4	875/22500	822/57296	677/149952	428/349184	196/897024

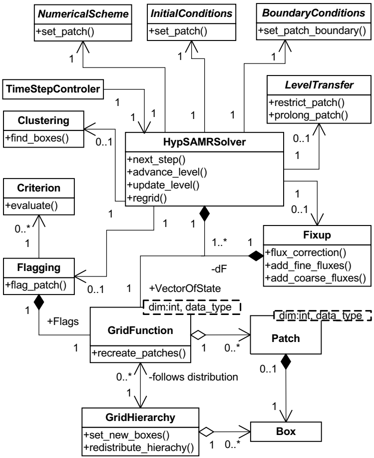
Comparison of number of cells and grids in DAGH and AMROC

The Virtual Test Facility

- Implements all described algorithms beside multigrid methods
- AMROC V2.0 plus solid mechanics solvers
- Implements explicit SAMR with different finite volume solvers
- Embedded boundary method, FSI coupling
- $\sim 430,000$ lines of code total in C++, C, Fortran-77, Fortran-90
- autoconf / automake environment with support for typical parallel high-performance system
- http://www.cacr.caltech.edu/asc
- [Deiterding et al., 2006][Deiterding et al., 2007]

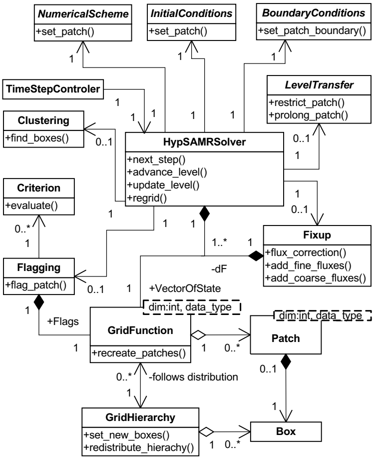
UML design of AMROC

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



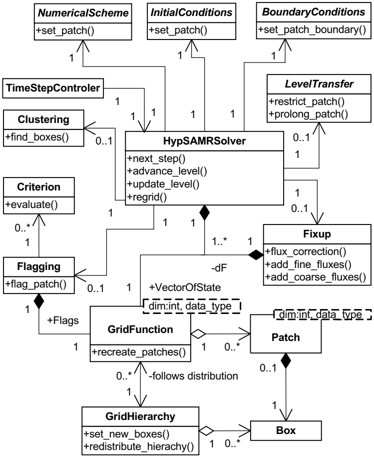
UML design of AMROC

- 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



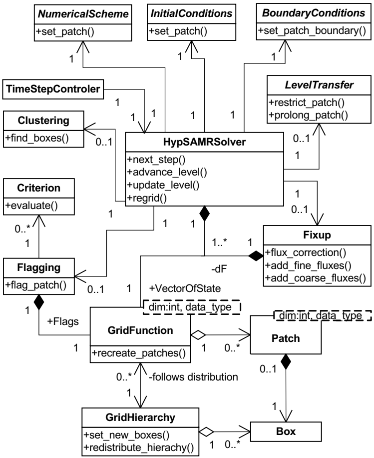
UML design of AMROC

- 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 (with F77 interfaces) provided for standard simulations



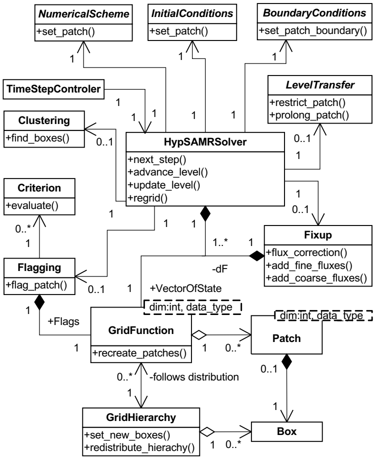
UML design of AMROC

- 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 (with F77 interfaces) provided for standard simulations
- Standard simulations require only linking to F77 functions for initial and boundary conditions, source terms. No C++ knowledge required



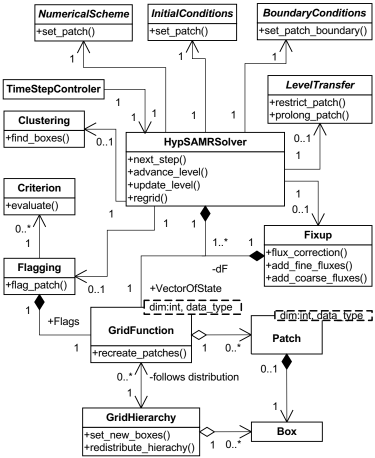
UML design of AMROC

- 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 (with F77 interfaces) provided for standard simulations
- Standard simulations require only linking to F77 functions for initial and boundary conditions, source terms. No C++ knowledge required
- Interface mimics Clawpack



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 (with F77 interfaces) provided for standard simulations
- Standard simulations require only linking to F77 functions for initial and boundary conditions, source terms. No C++ knowledge required
- Interface mimics Clawpack
- 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

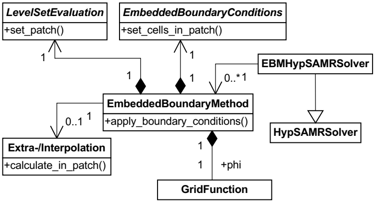
- Index coordinate system based on $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin the hierarchy
- `Box<dim>`, `BoxList<dim>` class that define rectangular regions $G_{m,l}$ by `lowerleft`, `upperright`, `stepsize` and specify topological operations $\cap$, $\cup$, $\setminus$
- `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

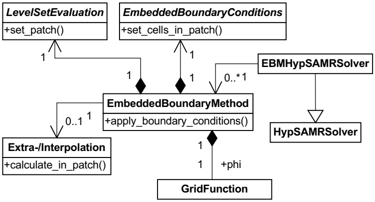
Embedded boundary method / FSI coupling

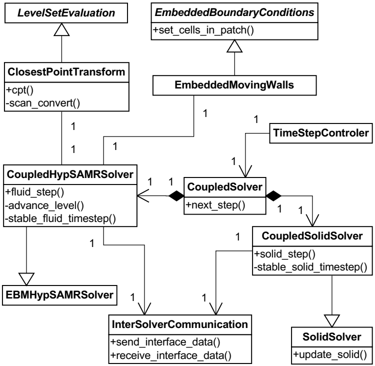
- Multiple independent EmbeddedBoundaryMethod objects possible
- Specialization of GFM boundary conditions, level set description in scheme-specific F77 interface classes



Embedded boundary method / FSI coupling

- Multiple independent EmbeddedBoundaryMethod objects possible
- Specialization of GFM boundary conditions, level set description in scheme-specific F77 interface classes





- Coupling algorithm implemented in further derived HypSAMRSolver class
- Level set evaluation always with CPT algorithm
- Parallel communication through efficient non-blocking communication module
- Time step selection for both solvers through CoupledSolver class

Parallelized construction of space-filling curve

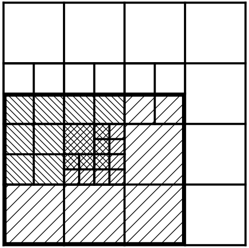
Computation of space filling curve

- Partition-Init

Parallelized construction of space-filling curve

Computation of space filling curve

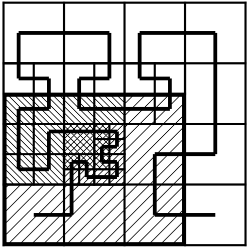
- Partition-Init
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0



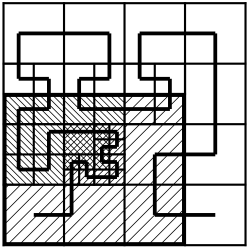
Parallelized construction of space-filling curve

Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid



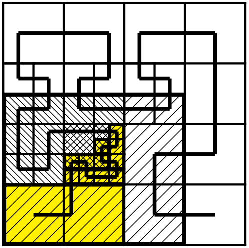
Parallelized construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid
- Partition-Calc
  1. Compute entire workload and new work for each processor

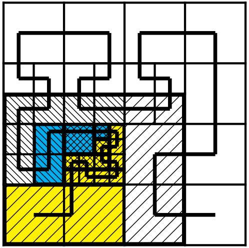
Parallelized construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid
- Partition-Calc
  1. Compute entire workload and new work for each processor
  2. Go sequentially through SFC-ordered list of partitioning units and assign units to processors, refine partition if necessary and possible

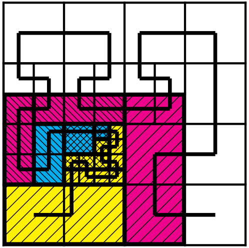
Parallelized construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid
- Partition-Calc
  1. Compute entire workload and new work for each processor
  2. Go sequentially through SFC-ordered list of partitioning units and assign units to processors, refine partition if necessary and possible

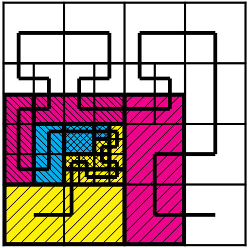
Parallelized construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid
- Partition-Calc
  1. Compute entire workload and new work for each processor
  2. Go sequentially through SFC-ordered list of partitioning units and assign units to processors, refine partition if necessary and possible

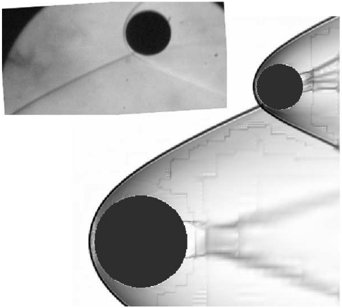
Parallelized construction of space-filling curve

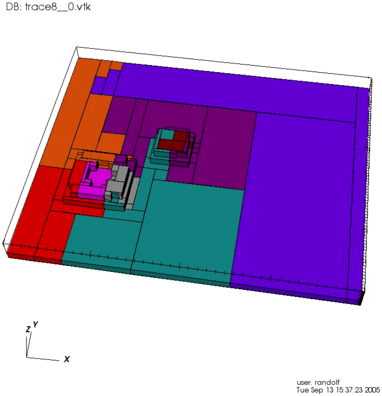


Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid
- Partition-Calc
  1. Compute entire workload and new work for each processor
  2. Go sequentially through SFC-ordered list of partitioning units and assign units to processors, refine partition if necessary and possible
- Ensure scalability of Partition-Init by creating SFC-units strictly local
- Currently still use of `MPI_allgather()` to create globally identical input for Partition-Calc (can be a bottleneck for weak scalability)

Partitioning example





- Cylinders of spheres in supersonic flow
- Predict force on secondary body
- Right: 200x160 base mesh, 3 Levels, factors 2,2,2, 8 CPUs

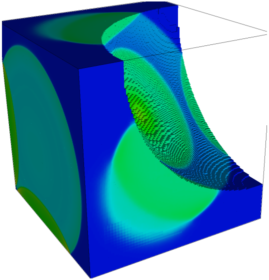
[Laurence et al., 2007]

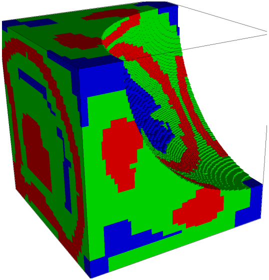
First performance assessment

- Test run on 2.2 GHz AMD Opteron quad-core cluster connected with Infiniband
- Cartesian test configuration
- Spherical blast wave, Euler equations, 3rd order WENO scheme, 3-step Runge-Kutta update
- AMR base grid 64^3, $r_{1,2} = 2$, 89 time steps on coarsest level
- With embedded boundary method: 96 time steps on coarsest level
- Redistribute in parallel every 2nd base level step
- Uniform grid $256^3 = 16.8 \cdot 10^6$ cells

Level	Grids	Cells
0	115	262,144
1	373	1,589,808
2	2282	5,907,064

Grid and cells used on 16 CPUs





Cost of SAMR and ghost-fluid method

- Flux correction is negligible
- Clustering is negligible (already local approach). For the complexities of a scalable global clustering algorithm see [Gunney et al., 2007]

CPUs	16	32	64
Time per step	32.44s	18.63s	11.87s
Uniform	59.65s	29.70s	15.15s
Integration	73.46%	64.69%	50.44%
Flux Correction	1.30%	1.49%	2.03%
Boundary Setting	13.72%	16.60%	20.44%
Regridding	10.43%	15.68%	24.25%
Clustering	0.34%	0.32%	0.26%
Output	0.29%	0.53%	0.92%
Misc.	0.46%	0.44%	0.47%

Cost of SAMR and ghost-fluid method

- Flux correction is negligible
- Clustering is negligible (already local approach). For the complexities of a scalable global clustering algorithm see [Gunney et al., 2007]
- Costs for GFM constant around $\sim 36\%$
- Main costs: `Regrid(l)` operation and ghost cell synchronization

CPUs	16	32	64
Time per step	32.44s	18.63s	11.87s
Uniform	59.65s	29.70s	15.15s
Integration	73.46%	64.69%	50.44%
Flux Correction	1.30%	1.49%	2.03%
Boundary Setting	13.72%	16.60%	20.44%
Regridding	10.43%	15.68%	24.25%
Clustering	0.34%	0.32%	0.26%
Output	0.29%	0.53%	0.92%
Misc.	0.46%	0.44%	0.47%

CPUs	16	32	64
Time per step	43.97s	25.24s	16.21s
Uniform	69.09s	35.94s	18.24s
Integration	59.09%	49.93%	40.20%
Flux Correction	0.82%	0.80%	1.14%
Boundary Setting	19.22%	25.58%	28.98%
Regridding	7.21%	9.15%	13.46%
Clustering	0.25%	0.23%	0.21%
GFM Find Cells	2.04%	1.73%	1.38%
GFM Interpolation	6.01%	10.39%	7.92%
GFM Overhead	0.54%	0.47%	0.37%
GFM Calculate	0.70%	0.60%	0.48%
Output	0.23%	0.52%	0.74%
Misc.	0.68%	0.62%	0.58%

AMROC scalability tests

Basic test configuration

- Spherical blast wave, Euler equations, 3D wave propagation method
- AMR base grid 32^3 with $r_{1,2} = 2, 4$. 5 time steps on coarsest level
- Uniform grid $256^3 = 16.8 \cdot 10^6$ cells, 19 time steps
- Flux correction deactivated
- No volume I/O operations
- Tests run IBM BG/P (mode VN)

AMROC scalability tests

Basic test configuration

- Spherical blast wave, Euler equations, 3D wave propagation method
- AMR base grid 32^3 with $r_{1,2} = 2, 4$. 5 time steps on coarsest level
- Uniform grid $256^3 = 16.8 \cdot 10^6$ cells, 19 time steps
- Flux correction deactivated
- No volume I/O operations
- Tests run IBM BG/P (mode VN)

Weak scalability test

- Reproduction of configuration each 64 CPUs
- On 1024 CPUs: $128 \times 64 \times 64$ base grid, $> 33,500$ Grids, $\sim 61 \cdot 10^6$ cells, uniform $1024 \times 512 \times 512 = 268 \cdot 10^6$ cells

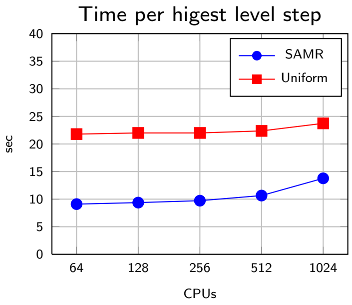
Level	Grids	Cells
0	606	32,768
1	575	135,312
2	910	3,639,040

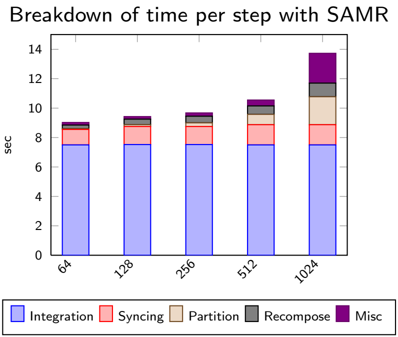
Strong scalability test

- $64 \times 32 \times 32$ base grid, uniform $512 \times 256 \times 256 = 33.6 \cdot 10^6$ cells

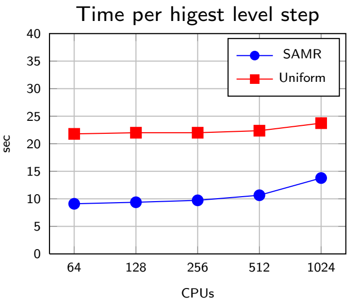
Level	Grids	Cells
0	1709	65,536
1	1735	271,048
2	2210	7,190,208

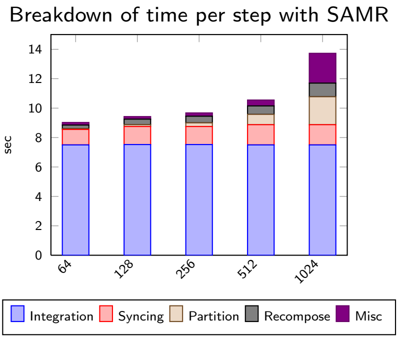
Weak scalability test





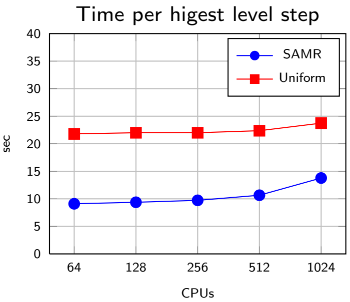
Weak scalability test

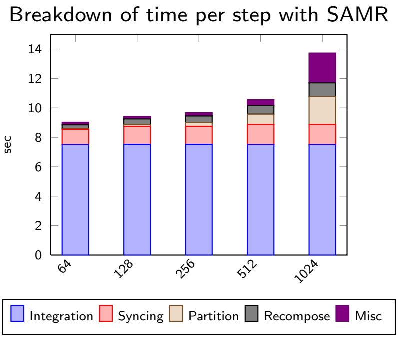




- Costs for Syncing basically constant

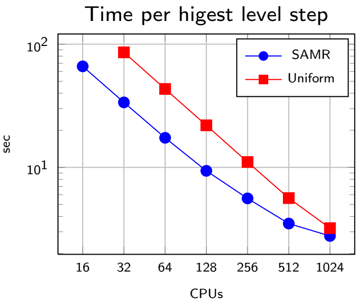
Weak scalability test

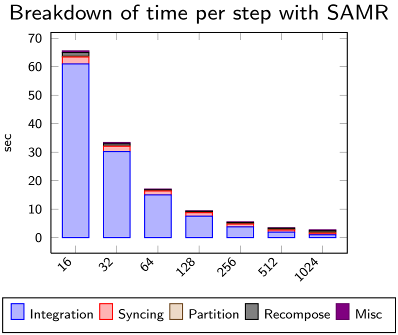




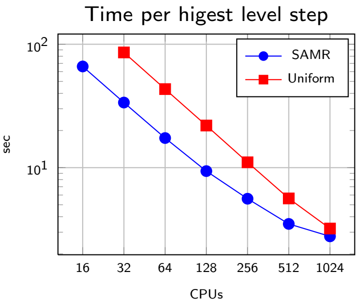
- Costs for Syncing basically constant
- Partitioning, Recompose, Misc (origin not clear) increase
- 1024 required usage of -DUAL option due to usage of global lists data structures in Partition-Calc and Recompose

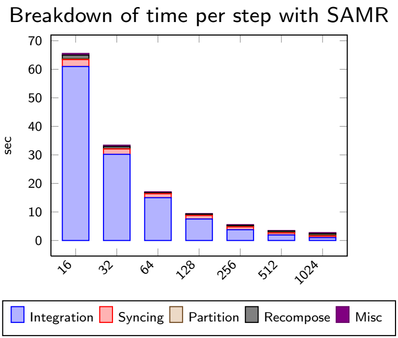
Strong scalability test





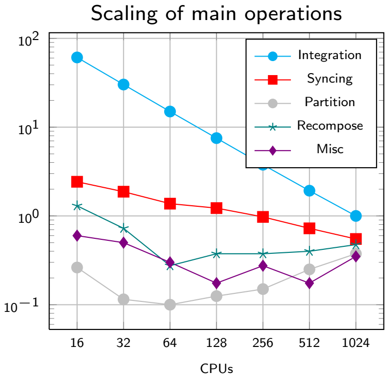
Strong scalability test

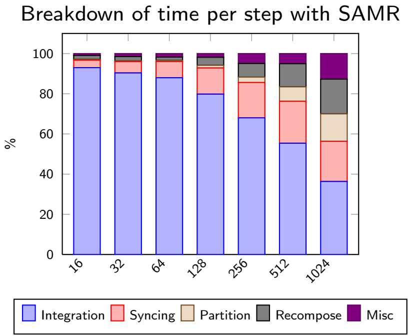




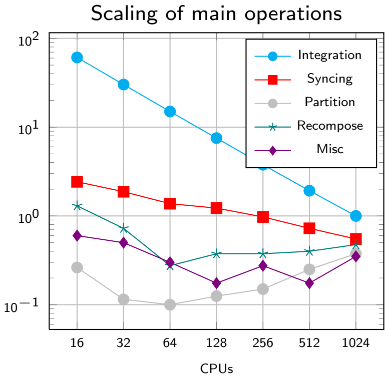
- Uniform code has basically linear scalability (explicit method)
- SAMR visibly looses efficiency for > 512 CPU, or $15,000$ finite volume cells per CPU

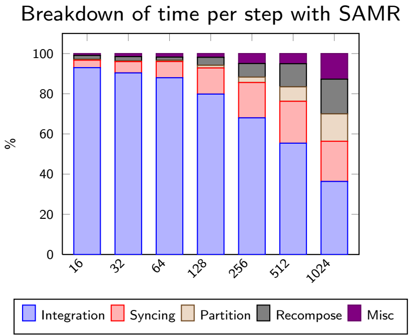
Strong scalability test - II





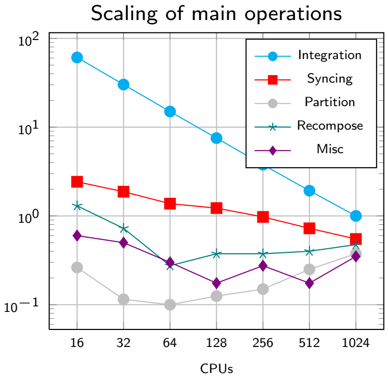
Strong scalability test - II

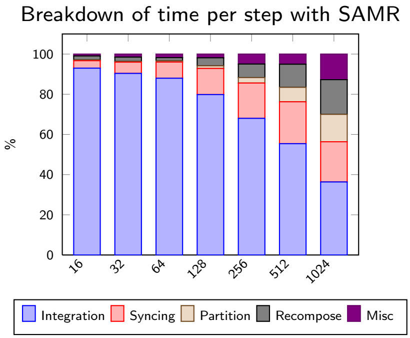




- Perfect scaling of Integration, reasonable scaling of Syncing

Strong scalability test - II

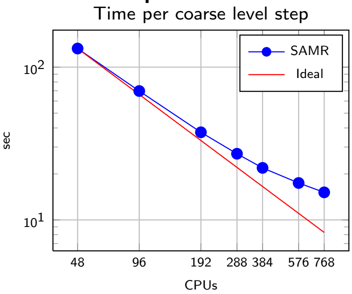




- Perfect scaling of Integration, reasonable scaling of Syncing
- Strong scalability of Partition needs to be addressed (eliminate global lists)

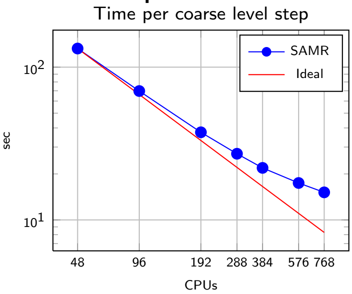
Strong scalability test - Train side wind computation

- Computation is restarted from disk checkpoint at $t = 0.526408\,\mathrm{s}$.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M) vs. 19.66 billion (uniform).



Strong scalability test - Train side wind computation

- Computation is restarted from disk checkpoint at $t = 0.526408\,\mathrm{s}$.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M) vs. 19.66 billion (uniform).

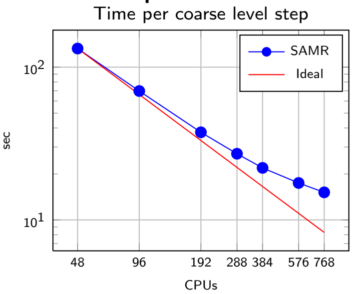


Time in % spent in main operations

Cores	48	96	192	288	384	576	768
Time per step	132.43s	69.79s	37.47s	27.12s	21.91s	17.45s	15.15s
Par. Efficiency	100.0	94.88	88.36	81.40	75.56	63.24	54.63
LBM Update	5.91	5.61	5.38	4.92	4.50	3.73	3.19
Regridding	15.44	12.02	11.38	10.92	10.02	8.94	8.24
Partitioning	4.16	2.43	1.16	1.02	1.04	1.16	1.34
Interpolation	3.76	3.53	3.33	3.05	2.83	2.37	2.06
Sync Boundaries	54.71	59.35	59.73	56.95	54.54	52.01	51.19
Sync Fixup	9.10	10.41	12.25	16.62	20.77	26.17	28.87
Level set	0.78	0.93	1.21	1.37	1.45	1.48	1.47
Interp./Extrap.	3.87	3.81	3.76	3.49	3.26	2.75	2.39
Misc	2.27	1.91	1.79	1.67	1.58	1.38	1.25

Strong scalability test - Train side wind computation

- Computation is restarted from disk checkpoint at $t = 0.526408\,\mathrm{s}$.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M) vs. 19.66 billion (uniform).
- Portions for parallel communication quite considerable (4 ghost cells still used).



Time in % spent in main operations

Cores	48	96	192	288	384	576	768
Time per step	132.43s	69.79s	37.47s	27.12s	21.91s	17.45s	15.15s
Par. Efficiency	100.0	94.88	88.36	81.40	75.56	63.24	54.63
LBM Update	5.91	5.61	5.38	4.92	4.50	3.73	3.19
Regridding	15.44	12.02	11.38	10.92	10.02	8.94	8.24
Partitioning	4.16	2.43	1.16	1.02	1.04	1.16	1.34
Interpolation	3.76	3.53	3.33	3.05	2.83	2.37	2.06
Sync Boundaries	54.71	59.35	59.73	56.95	54.54	52.01	51.19
Sync Fixup	9.10	10.41	12.25	16.62	20.77	26.17	28.87
Level set	0.78	0.93	1.21	1.37	1.45	1.48	1.47
Interp./Extrap.	3.87	3.81	3.76	3.49	3.26	2.75	2.39
Misc	2.27	1.91	1.79	1.67	1.58	1.38	1.25

References I

[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.

[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 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.

[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.

References II

[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.

[Gunney et al., 2007] Gunney, B. T., Wissink, A. M., and Hysoma, D. A. (2007). Parallel clustering algorithms for structured AMR. *J. Parallel and Distributed Computing*, 66(11):1419–1430.

[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.

[Laurence et al., 2007] Laurence, S. J., Deiterding, R., and Hornung, H. G. (2007). Proximal bodies in hypersonic flows. *J. Fluid Mech.*, 590:209–237.

[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 III

[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.